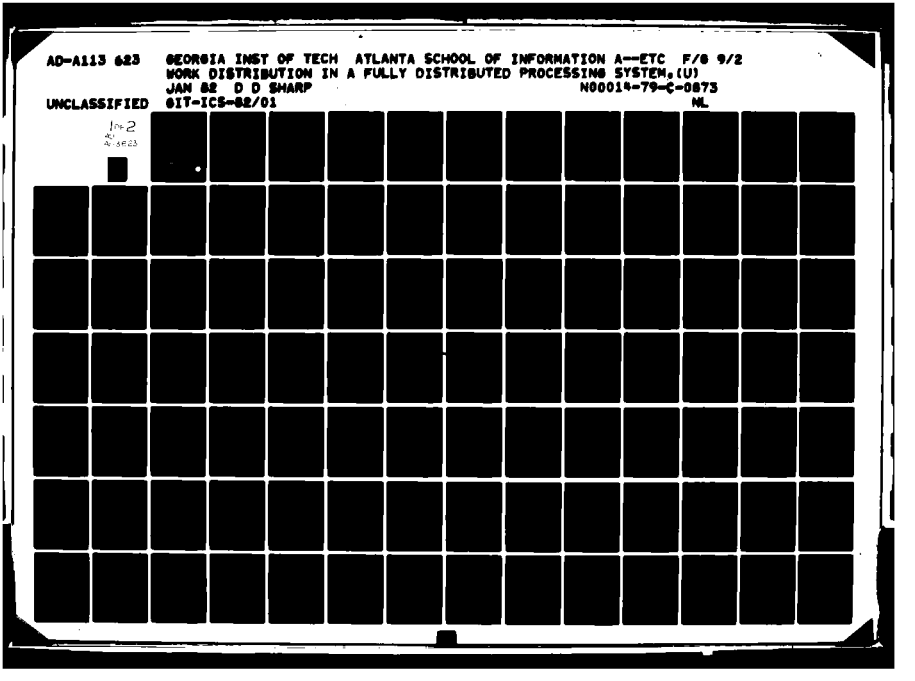

AD-A113 623 GEORGIA INST OF TECH ATLANTA SCHOOL OF INFORMATION A--ETC F/G 9/2
WORK DISTRIBUTION IN A FULLY DISTRIBUTED PROCESSING SYSTEM.(U)
JAN 82 D D SHARP N00014-79-C-0873
UNCLASSIFIED GIT-ICS-82/01 NL
1 of 2
AD A113623

(14)

AD _____

TECHNICAL REPORT
GIT–ICS–82/01

WORK DISTRIBUTION IN A FULLY DISTRIBUTED PROCESSING SYSTEM

By

Donald D. Sharp, Jr.

Prepared for

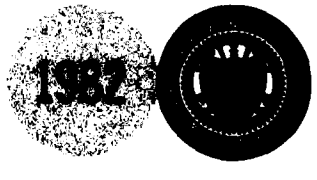
OFFICE OF NAVAL RESEARCH
800 N. QUINCY STREET
ARLINGTON, VA. 22217

Under

Contract No. N00014–79–C–0873
GIT Project No. G36–643

January 1982

GEORGIA INSTITUTE OF TECHNOLOGY
A UNIT OF THE UNIVERSITY SYSTEM OF GEORGIA
SCHOOL OF INFORMATION AND COMPUTER SCIENCE
ATLANTA, GEORGIA 30332



THE RESEARCH PROGRAM IN
FULLY DISTRIBUTED PROCESSING SYSTEMS

WORK DISTRIBUTION IN A FULLY DISTRIBUTED PROCESSING SYSTEM

TECHNICAL REPORT
GIT-ICS-82/01

Donald D. Sharp, Jr.

January, 1982

Office of Naval Research
800 N. Quincy St.
Arlington, VA 22217

Contract Number N00014-79-C-0873
GIT Project Number G36-643

The Georgia Tech Research Program in
Fully Distrbuted Processing Systems
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

THE VIEW, OPINIONS, AND/OR FINDINGS CONTAINED IN THIS REPORT ARE THOSE OF THE AUTHOR AND SHOULD NOT BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE NAVY POSITION, POLICY, OR DECISION, UNLESS SO DESIGNATED BY OTHER DOCUMENTATION.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER GIT-ICS-82/01	2. GOVT ACCESSION NO. AD-A113623	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Work Distribution in a Fully Distributed Processing System		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER GIT-ICS-82/01
7. AUTHOR(s) Donald D. Sharp, Jr.		8. CONTRACT OR GRANT NUMBER(s) N00014-79-C-0873
9. PERFORMING ORGANIZATION NAME AND ADDRESS School of Information and Computer Science Georgia Institute of Technology Atlanta, Georgia 30332		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research (ONR) 800 N. Quincy St. Arlington, Virginia 22217		12. REPORT DATE January 1982 13. NUMBER OF PAGES 147 + viii
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) same as item 11		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

same

18. SUPPLEMENTARY NOTES

The view, opinions, and/or findings contained in this report are those of the author and should not be construed as an official Department of the Navy position, policy, or decision, unless so designated by other documentation.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Fully Distributed Processing Systems
Work Distribution

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

In a fully distributed data processing system, work requests may be generated by either users or executing processes. The servicing of each work request requires the use of a set or collection of system resources. Multiple copies of the resources required may be present within the overall system. The problem is to select the specific instances of these resources (needed for the execution of each process generated from the work request) so that average user response time will be minimized and system throughput will be

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

maximized. This thesis investigates the work distribution problem by using a simulation model to analyze the performance of three work distribution algorithms in test cases which simulated various network conditions. The first work distribution algorithm attempts to minimize communications between the network nodes, the second algorithm tries to balance the processing load on the processors at each node, and the third algorithm is a combination of the other two.

Two sets of experiments were conducted. The first set compared average user response time for the three algorithms for three examples of work request, three network topologies, and two levels of file redundancy. The second set of experiments compared the average user response time for the different algorithms at different communication bandwidths. The effects of different network topologies and degrees of file redundancy were also tested in the second set of experiments.

The results of the simulation experiment showed that the algorithm that attempts to minimize communications was better than the other two algorithms in terms of minimizing average user response time under the specific conditions tested. The performance of this algorithm was especially good with low bandwidths, with a global bus topology, and with multiple file copies.

ABSTRACT

In a fully distributed data processing system, work requests may be generated by either users or executing processes. The servicing of each work request requires the use of a set or collection of system resources. Multiple copies of the resources required may be present within the overall system. The problem is to select the specific instances of these resources (needed for the execution of each process generated from the work request) so that average user response time will be minimized and system throughput will be maximized. This thesis investigates the work distribution problem by using a simulation model to analyze the performance of three work distribution algorithms in test cases which simulated various network conditions. The first work distribution algorithm attempts to minimize communications between the network nodes, the second algorithm tries to balance the processing load on the processors at each node, and the third algorithm is a combination of the other two.

Two sets of experiments were conducted. The first set compared average user response time for the three algorithms for three examples of work request, three network topologies, and two levels of file redundancy. The second set of experiments compared the average user response time for the different algorithms at different communication bandwidths. The effects of different network topologies and degrees of file redundancy were also tested in the second set of experiments.

The results of the simulation experiment showed that the algorithm that attempts to minimize communications was better than the other two algorithms in terms of minimizing average user response time under the specific conditions tested. The performance of this algorithm was especially good with low bandwidths, with a global bus topology, and with multiple file copies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

SECTION 1

Distributed data processing systems are currently being studied by researchers and prospective users because of their potential for improvements in system performance, reliability, and resource sharing; and this area has become the focus for a major research program at the Georgia Institute of Technology. Since the number of definitions of distributed systems is quite large, we at Georgia Tech have identified our specific area of interest as "Fully Distributed Processing Systems" (FDPS) as described in papers by Enslow [Ensl78], and Enslow and Saponas [Ensl81a].

One of the major distinguishing properties of a fully distributed processing system (FDPS) is the extremely loose physical and logical coupling of all resources. The FDPS is composed of a multiplicity of general purpose computers connected together in a network by means of communication links. Unlike the tight physical coupling of multiprocessors which share primary memory, the computers in an FDPS can communicate with each other only by means of messages sent across these communication links. Furthermore, loose coupling implies that the speed of the communication links is at least one order of magnitude slower than the speed of the processors [Peeb78, Ston78b] and possibly much less.

A second, and extremely important, property of an FDPS is the control philosophy of "cooperative autonomy". In order to provide modularity and reliability, control of the network must be distributed and decentralized rather than centralized. Loose physical coupling demands a "two-party message transfer protocol" in which both the sending party and the receiving party must actively participate and cooperate in message transmission. Loose logical coupling in an FDPS means that there are no fixed master-slave relationships. Thus, each node of the network operates autonomously but according to a unified set of policies, the overall network operating system policies. Since local autonomy implies that each node can refuse a request for service [Clar80, Ensl78], anarchy could result except for the fact that each node does adhere to the same set of policies. It is envisioned that a portion of the "network" operating system will operate autonomously at each

node and will provide "network" functions and services while communicating with the other nodes. This will allow each node to operate independently and asynchronously, but in compliance with the global policies.

Another important property of the FDPS is "system transparency". The system is transparent to the user if the system can provide all of the services which the user requires without the user having to identify the specific name or location of the physical resources to be used to provide those services. Because it lacks this system transparency as well as other features discussed above, the computers attached to ARPANET [Robe73] would not be considered a "fully distributed processing system".

The notions of system transparency and autonomy of operation place a very large burden on the network operating system. The network operating system of a fully distributed system must manage all resources in a global manner and respond to all user requests for named services. It is this feature of fully distributed systems which will challenge the experimenters and system designers in the 1980's.

This research investigates the problem of developing an efficient method for assigning units of work (tasks) to specific processors in a fully distributed data processing system in order to maximize certain measures of system performance. Issues involving system reliability will not be investigated nor will issues involving the initial (static) placement of resources. This chapter will discuss the problem to be solved and some of the simplifying assumptions which were made to define the scope of this research. Chapter Two provides an historical perspective and summary of the current state of the art of other research in this area. Chapter Three will present the experimental method employed. Chapter Four presents the experimental results, and Chapter Five discusses the experimental findings and their significance. Chapter Six presents the conclusions supported by these experiments, and Chapter Seven describes areas for future research.

1.1 The Distributed System Environment

The fully distributed processing system (FDPS) is composed of N nodes connected together by communication links. All nodes can communicate with every other node, but not necessarily through a direct link. For the purposes

of this research, it is assumed that each node consists of one computer system consisting of a processor (CPU), primary memory, secondary memory (e.g., disk drives), user input/output terminals, and possibly other input/output devices such as a line printer or a plotter. The hardware that is simulated for each node is shown in Figure 1.

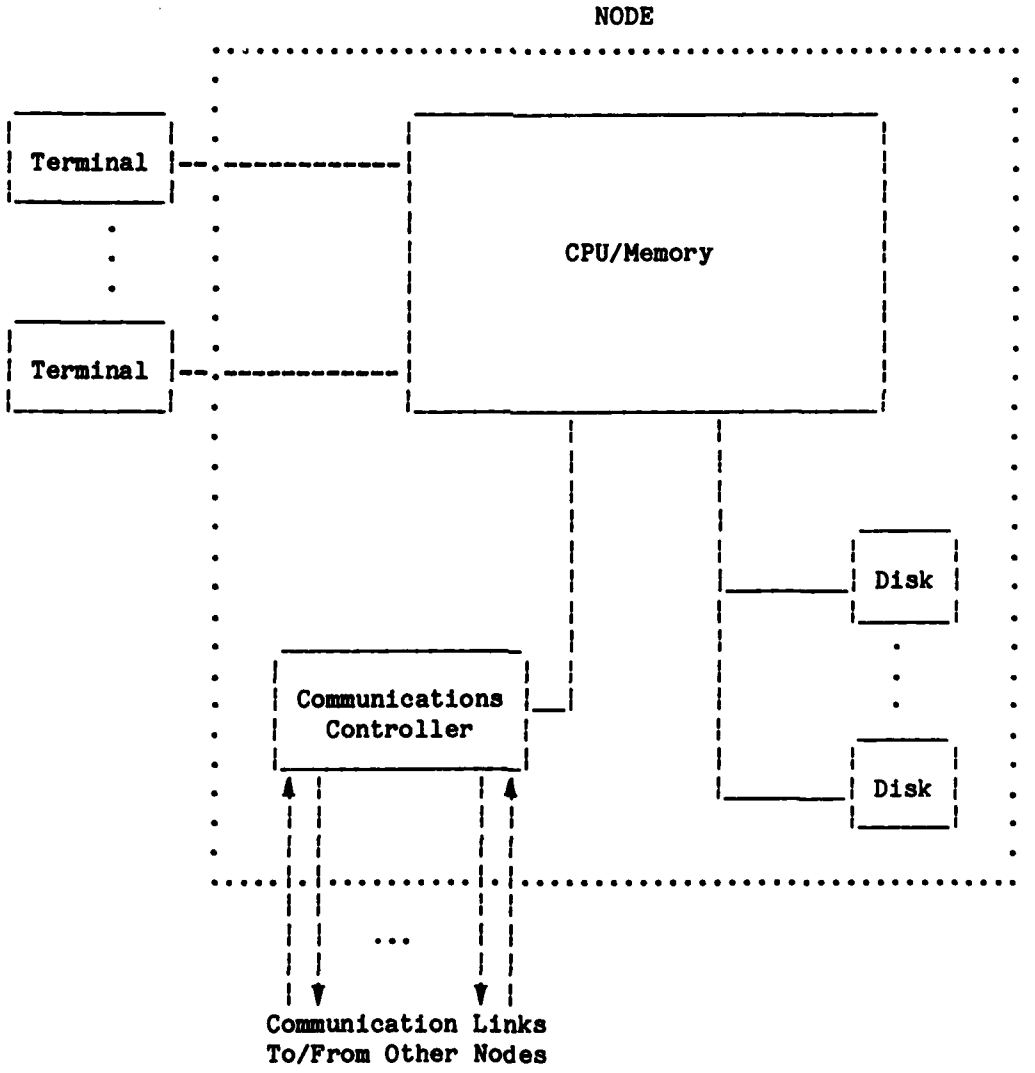


Figure 1. The Architecture Supported by the Simulator for Each Node

Other researchers [Fors80, Jone80, Wu80] have examined networks where each node consists of a cluster of physical processors. The difference between several physical processors at each node and several virtual processors (within a single time-shared or multiprogrammed machine) does not seem to be an important distinction when deciding whether to assign a process to a particular node. However, deciding which processor within a cluster of processors at a node to utilize is a different problem and is beyond the scope of the present research. The conclusions based on the results from this research should apply to both forms of organization.

The distributed systems considered in this initial research on work distribution and resource allocation consisted of identical, homogeneous processors. In this context, "homogeneous" does not necessarily mean the same thing as "identical". Two processors with different processing capability but identical instruction sets and architecture; two different models within the family of IBM Series 370 computers would be homogeneous. It is assumed in this experiment that all processors are identical.

The processor at each node is assumed to be capable of supporting several interactive users. Each processor receives work requests from those users as well as from executing processes (e.g., a request to spawn a new process). The execution of each work request requires the use of various system resources, some of which may have several identical or equally satisfactory copies at different network locations. The selection of specific system resources to service each work request is a function of the work distribution procedure. The result of the selection process is a proposal for which processes are to be executed on specific network processors. The selected processor(s) will schedule and execute the process(es) in the same manner that it would handle a locally-generated work request. The node at which the original work request originates will be called the "source node".

A portion of the operating system at each node is concerned primarily with network functions. The operating system accepts work requests from users and from active processes, determines resource availability, negotiates with other nodes to assign processes to specific nodes for execution, arranges for the transfer of data resources between nodes, and monitors the execution of each process. It is assumed that a knowledgeable user can specify explicitly

the assignment of processes to processors if desired.

All communication between processes is assumed to be by means of messages sent on the various communication links [Brit80, Macc80]. A link is a one-way communication path between a pair of nodes. All input/output operations (i.e., reads and writes) are likewise assumed to consist of send and receive commands. Messages are addressed to processes and are linked to processes by means of a named connector called a port [Akin78, Macc80, Mill76].

Each communication link has a fixed transmission rate (baud rate). In order to simplify the experimental model, no requirement for flow control [Gerl80, Klei80] is implemented. Therefore, link queues, message queues, and port queues are unlimited in size.

Many types of resources are of interest in the present research. Processors are assumed to exist at each node of the network. The specific devices (e.g., terminals, disk drives, printers) attached to these processors will, in general, not be an important consideration. Users are assumed to interact with the system by means of interactive terminals.

The placement and use of data files was an important element of this research. A program is a special kind of data file that can be executed. The terms "program" and "program module" will be used interchangeably to specify any piece of computer code capable of being executed. A process is simply an instantiation of a program. Data files and program files are considered to be passive resources because they are acted upon. Processes are considered to be active resources because they perform some actions.

Server processes are assumed to exist as part of the "network" operating system to support actions unique to a distributed system, such as remote data access based on a request from another node. Processes may spawn (create) other processes during their period of execution. The newly spawned process must then be assigned to a processor and scheduled like any other new user process. The migration of processes from node to node during execution will not be considered in this research. It is assumed that, if a process fails in the middle of its execution, it will be restarted from the beginning, usually after being reassigned to a different processor.

The sharing of data files (including programs) is a critical aspect of this research. Morgan and Levin [Morg77] define three levels of data sharing, with the highest level being the sharing of all data files and programs. In a distributed system, each data file may have several identical copies located at different nodes, in order to provide increased system reliability and facilitate data access. The problem of updating multiple copies of a data file, in a system with no centralized control, however, is still a key research question. Solutions to this update problem have been suggested by Thomas [Thom76], Ellis [Elli77], Garcia-Molina [Garc79a, Garc79b], and Gardarin and Chu [Gard80]. Bernstein and Goodman [Bern80] present a rather comprehensive survey of work in this area.

A second approach to data file placement is to partition the entire system data base so that each data file is located at one and only one node. This solves the problem of updating multiple copies of data files, but it may require remote data access or the copying of data files from node to node prior to data file usage. The determination of the number of copies of a data file and their placement is another major design problem in distributed systems that is not addressed here.

1.2 The Work Distribution Problem

The specific problem addressed in this research is the development and evaluation of several work distribution algorithms. The work distribution algorithms should assign executable processes to system processors in a manner that maximizes system performance. The most commonly stated performance goals for distributed systems are the minimization of (average) user response time and the maximization of total system throughput. Most reseachers and system designers would agree that these two goals are very important, but there seems to be no consensus as to which is the more important. Since these two goals are often conflicting, the present research will evaluate performance separately for each of these two goals.

Because of the wide range of possible distributed system configurations, resource placement decisions, and system workloads, a single work distribution algorithm is not expected to provide optimal performance under all conditions. It is therefore the goal of the present research to test several work distribution algorithms in order to determine under what conditions each one

might be more appropriate. The specific hypotheses to be tested in this experiment are as follows:

Null Hypothesis 1:

There is no significant difference in average user response time due to the use of any one of the three work distribution algorithms.
(mean.1 = mean.2 = mean.3)

Null Hypothesis 2:

There is no significant difference in average user response time between any one of the three work distribution algorithms and a base-line algorithm which moves no files.
(mean.1 = mean.0, mean.2 = mean.0, mean.3 = mean.0)

Null Hypothesis 3:

There is no significant difference in average user response time for any of the three work distribution algorithms when multiple file copies are available rather than single file copies.
(mean.1M = mean.1S, mean.2M = mean.2S, mean.3M = mean.3S)
(mean.1M = mean.0, mean.2M = mean.0, mean.3M = mean.0)

The work distribution problem is difficult because all of the resources required by the process to be executed may not be available at the same node. Suppose that a work request is received at Node 1 which requires a program which exists only at Node 2 and a data file which exists only at Node 3. Further, input and output are "tied down" to the user's terminal at Node 1. There are several ways to execute this process in the distributed system environment just described. Figures 2-5 show how task graphs might be constructed and resources assigned if the work distribution algorithm selected each of the nodes in a four node network example for the execution of the program described in the work request. Details of task graph construction can be found in [Ensl81a]. The examples in Figures 2-5 use the following notation:

[ ]	visible external reference(s)
(n)a	responsibility for a delegated from node n
a(n)	responsibility for a delegated to node n
a,b,...	lowercase letters indicate executable files
x,y,z	indicate data files or user terminals
a'	indicates a temporary copy of a file a

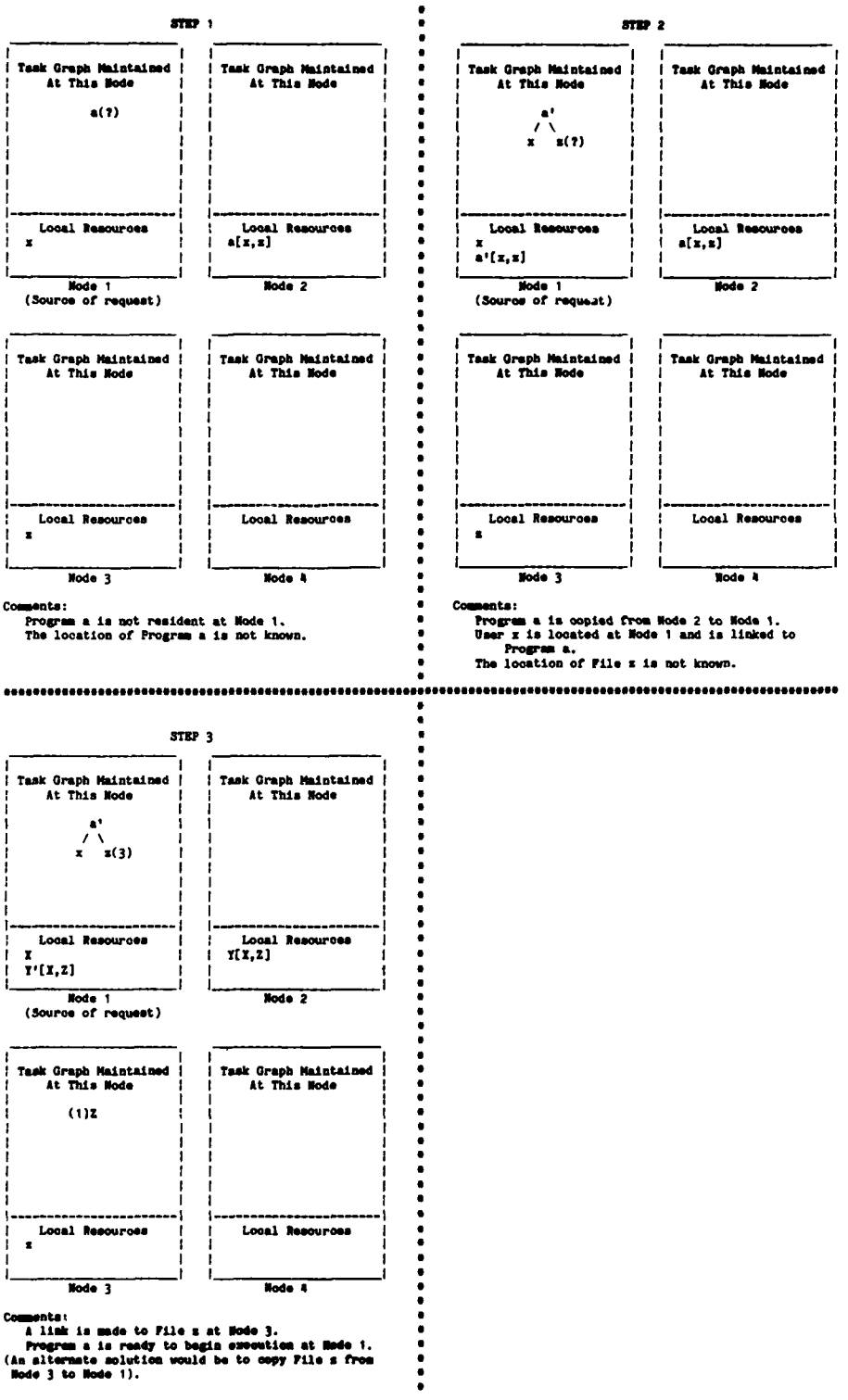


Figure 2. An Example of a Work Request Assignment

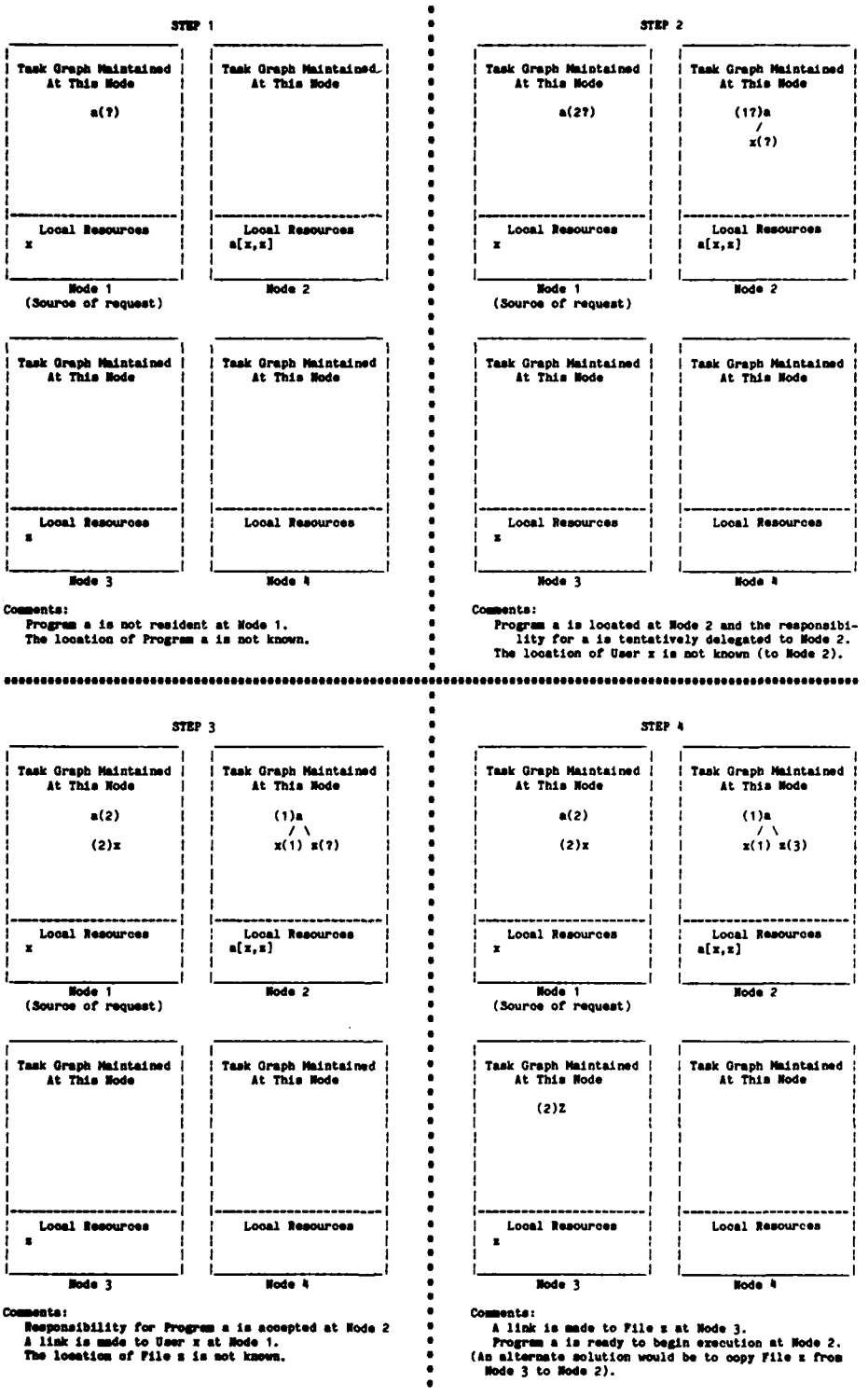


Figure 3. An Example of a Work Request Assignment

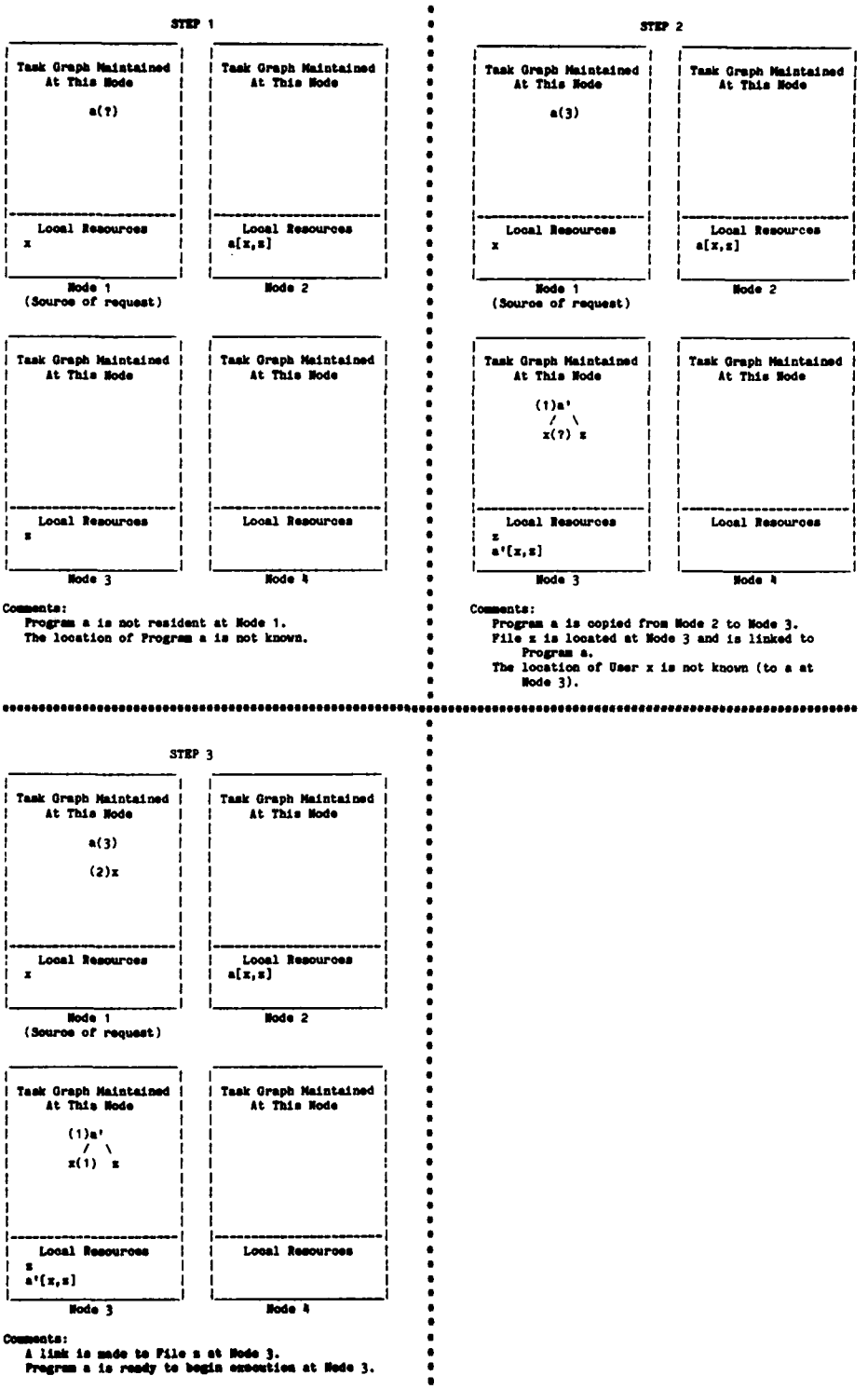


Figure 4. An Example of a Work Request Assignment

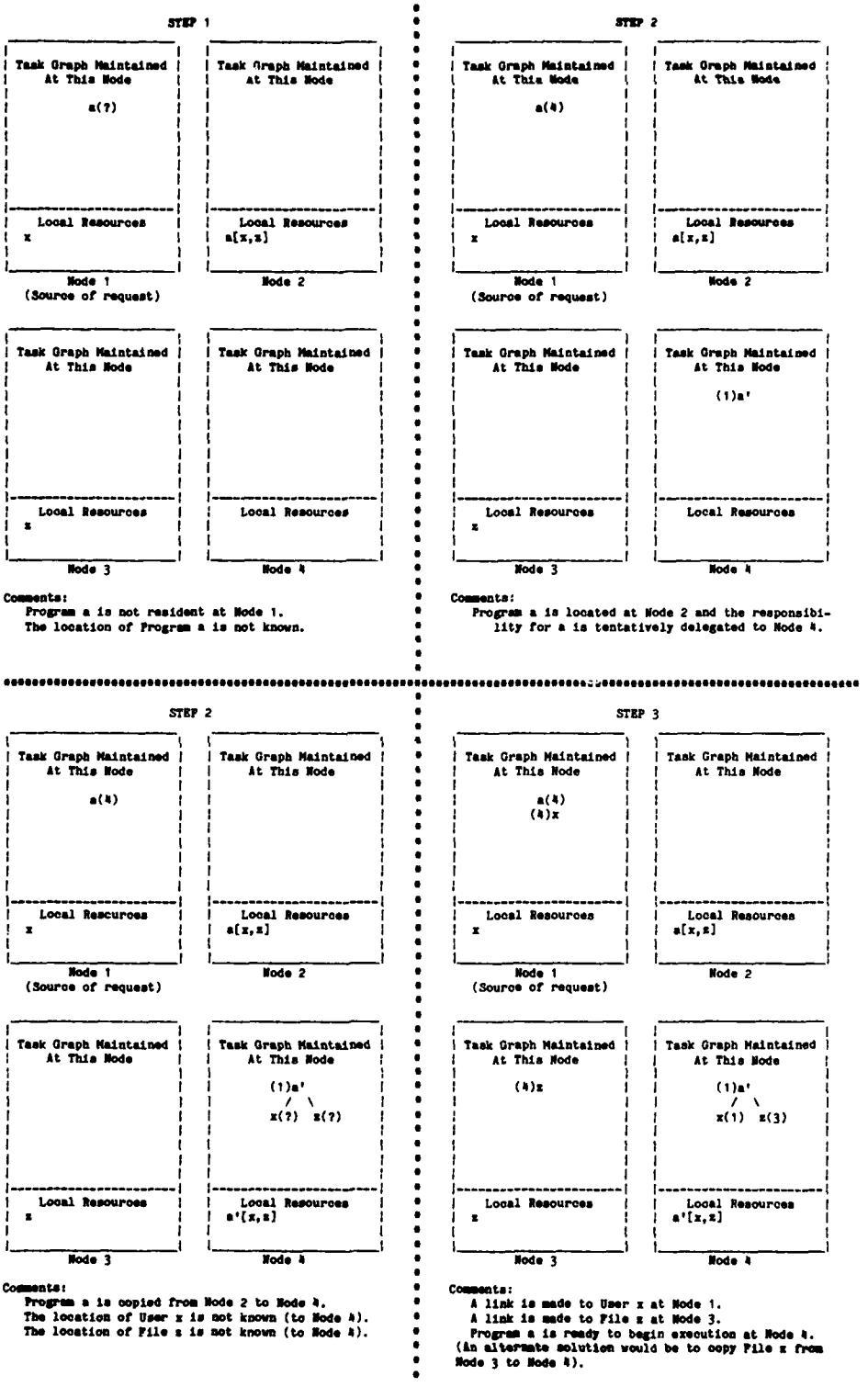


Figure 5. An Example of a Work Request Assignment

In Figure 2, the executable file is copied from Node 2 to Node 1 as might be the case if the message traffic that would be generated between x at Node 1 and the program a at Node 2 is larger than the traffic generated by copying the executable file, and the work distribution algorithm is trying to minimize communication traffic. In Figure 3, the executable file is executed where it is located and no file copying is performed. The situation in Figure 4 is similar to the one in Figure 2, but here the assumption is that a large amount of message traffic exists between the executable file at Node 2 and the data file at Node 3, so that copying the object file to Node 3 would minimize communication. In Figure 5, the executable file is moved to a node where none of the other files exist. This could be done to try to balance the load on the system if Node 4 is lightly loaded compared to other system nodes.

Even with this simple example it is easy to see that there are many options possible for assigning processes and files to system resources. With multiple input job streams, with multiple copies of files at some but not all nodes of the network, and with task sets composed of multiple processes capable of executing in parallel but with significant interprocess communication, the work distribution problem can become very difficult.

It also should be noted that the work distribution decision itself affects the amount of message traffic in the network and the distribution of that message traffic. This traffic results from information gathering and file movement decisions as well as the accessing of remote files and the transporting of "intermediate" results.

SECTION 2

The management of resources in a distributed system is a major topic of research. Jones and Schwarz [Jone80] make an important distinction between the two resource allocation decisions, placement decisions and assignment decisions. The physical positioning of resources (i.e., data, command, and program files) within the network constitutes the placement decision. The assignment decision concerns the allocation of executable processes to processors. The assignment decision is synonymous with work distribution in this dissertation; whereas, the placement decision is only indirectly related to the present research to the extent that the static placement of resources affects the dynamic assignment decisions to assign processes to processors. The placement decision is essentially a "static" or "long-term" decision, whereas the assignment decision is a dynamic "short-term" decision process performed only on a "demand" basis.

2.1 The Placement Decision

A significant amount of research has been directed toward the placement problem. Chu [Chu69, Chu73] proposed a mathematical programming solution to the problem of data file placement in a multi-computer system. Casey [Case72] presented a graph theoretic approach to the solution of the data file placement problem.

Morgan and Levin [Morg77, Levi75] expanded the previous research by considering the data dependencies present between programs and data files. They proposed a dynamic programming solution to the problem of positioning data files and programs in a computer network. Fisher and Hochbaum [Fish80] present a modification to the work of Morgan and Levin which uses a heuristic approach to solve the problem of positioning data files and programs. They point out that the more realistic models of placement problems are in the class of NP-hard problems, so that a heuristic approach is necessary. Another heuristic approach was presented by Mahmoud and Riordan [Mahm76]. All of the research mentioned above has attempted to optimize the placement of data files by comparing the cost of accessing the data with the cost of storing it. The

research of Fisher and Hochbaum was the first to give more than passing attention to the problem of determining the optimum number of copies for each data file.

The placement problem was examined in a somewhat larger context by several other researchers. Irani and Khabbaz [Iran79, Khab80] discussed the combined problem of data file allocation and communication network design. They presented a model that attempted to minimize the total cost of file storage and communication channels. A heuristic algorithm to solve the model was also presented. Chen and Akoka [Chen80] considered the simultaneous allocation of programs and data, processing power (processors), and communication line capacity, using a branch-and-bound integer programming technique. Chang and Liu [Chan79b] described a general model for distributed computer systems which investigated the problem of centralization versus dispersion of data. They believed that the system configuration problem could be formulated as a problem of determining transaction allocation, processor allocation, communication line allocation, and data file allocation. They presented a heuristic design procedure to generate system configurations for hierarchical (tree-like) computer systems. Finally, Buckles and Hardin [Buck79] described a static allocation method for distributing logical resources (data) within the physical configuration of the network. They presented a four-step, distributed system design approach which utilized a partitioning step, a mathematical programming step, and two simulation steps.

2.2 The Assignment Decision

Less research has been directed toward solving the assignment problem than toward the placement problem. Stone [Ston77] investigated the problem of process assignment in a distributed computing system composed of two heterogeneous processors. He showed how a commodity flow model can find an assignment of program modules (processes) to processors that minimizes the total absolute running time of a program (i.e., the total of processor time and communication time). His model assumed that the processing time for each process (on each of the processors) was known and fixed, and that the interprocess communication time between modules was also known and fixed.

Stone gave an example where a set of serially executable processes was divided between the two processors, but only because one process could run on

only one of the processors and another process could run only on the other. In a more general case where all processes are free to run on either processor, the solution would be to execute all processes on one processor (the faster of the two). If the processors were identical, then Stone's algorithm would produce two solutions (all processes executing on one processor or the other) and the algorithm can give no assistance in selecting between these two possible assignments.

Stone's 1978 paper [Ston78a] developed the concept of critical load factors. He calculated a critical load factor for every program module, so that, when the actual load on a processor was below the critical factor, the module was assigned to that processor; otherwise, it was assigned to the other processor. This approach opened the possibility of doing dynamic process assignments in real time. This solution was also much simpler than performing the maximum flow computation with a minimum cost cutset. However, this approach is restricted to serial execution of processes where each has a known computation time. Further, it assumes that the actual cost of a reassignment is negligible (e.g., copies of every process are on both processors).

Rao, Stone, and Hu [Rao79] examined the problem of minimum cost assignment of processes to two processors when one processor has limited memory and the other has unlimited memory, as is found in many host-satellite systems. Bokhari [Bokh79] further explored the two-node dynamic assignment model presented by Stone, but included the cost of dynamically reassigning a process from one processor to another during the course of program execution.

Stone and Bokhari [Ston78b] generalized Stone's basic algorithm for a three-processor network. The authors stated that the assignment problem for four or more processors is in the class of NP-hard problems for which an efficient solution may not even exist. Wu and Liu [Wu80] presented a solution for the k-processor ($k > 2$) network which yielded an optimal solution for the special case of a tree-like network. Chow and Kohler [Chow79] described in very general terms some basic queueing models for achieving load balancing in a tightly-coupled system of heterogeneous processors. They also presented a recursive technique for analyzing the two-processor system. The objective of their models was to reduce the average job turnaround time by balancing the workload among the processors.

Casey and Shelness [Case77a, Case77b], while investigating domain structures and capability mechanisms for distributed systems, developed a "best site calculation" for selecting the location of a domain (i.e., an executable process, its associated data, and a virtual processor segment containing the information required to represent and manage a virtual processor). The best site calculation is a two-stage procedure. The initial phase of the calculation considers only three sites (nodes) as possible candidates for domain assignment: (1) the node containing the program module (process) to be executed, (2) the node containing the virtual processor segment (i.e., the source node where the request for process execution originated), or (3) the node containing local (unshared) data associated with the process. The site selection calculation is performed at the node where the program module (process) to be executed is located.

In general the site is chosen to minimize the communication load due to the movement of all necessary resources to one location. However, the workload at each node and the available (primary) memory at each node is also considered, as well as the location of other copies of the program file to be executed. This calculation tended to balance the load, to free memory at these three sites, and to aggregate all instantiations of a single process at one site. Information about the status of other processing sites was propagated by adding it to each message in the system. Each node maintained a table of the free memory and present workload at every node.

The second phase of the calculation is performed at the location of the "best" site chosen in phase one. In this phase, all nodes were considered, so that load balancing could be improved. If some other node is substantially less busy than the node chosen in phase one, then that less busy site is nominated as "best" site. While performing this second calculation, it was recognized that some domains could not be moved because they were "tied down" to drive a peripheral device attached to a particular node. Executable processes (domains) can be queued in this model if all resources needed are not presently available. In addition, the authors stated that deadlock could occur if the system becomes overloaded.

Chu and his associates [Chu80] attempted to summarize and evaluate much of the previous work presented above. They concluded that the assignment

problem is a critical issue in the design of distributed systems, and it is a problem that is far from being completely solved. Forsdick and his associates [Fors80] supported this conclusion and stated that "almost no relevant work has been done toward the aim of managing distributed resources to achieve global objectives". They further stated that "meaningful evaluation of the effectiveness of ... global resource management are strongly dependent on the application and the architecture on which the application is implemented".

2.3 Limitations of Previous Research

Most of the previous research on the work distribution problem by Stone and his associates [Bokh79, Rao79, Ston78a, Ston78b, Ston79] is limited in scope because only serial programs are considered (no parallel execution of tasks within a program), processing times and communication times are assumed to be fixed and known precisely (no variable delays in process queues or message queues), and multiple solutions are common. Also, load balancing is not an objective of these models, as can be seen from solutions which assign all processes to one of two available processors.

The graph theoretic approach of Stone and his associates quickly becomes computationally intractable when extended to even a moderate number of processors [Chu80]. Stone and Bokhari [Ston78b] also question the stability of their own algorithm. Kratzer and Hammerstrom [Krat80] showed that the general assignment problem is NP-complete. A more serious limitation to the graph theoretic approach when applied to fully distributed systems is the inability of the algorithm to account for the impact of queueing delays on system throughput [Chan79b]. This approach assumes a single job stream; whereas, a fully distributed system may have multiple and independent job streams and multiple assignment decisions taking place in parallel, introducing complex queueing delays. Queueing models for tightly coupled systems were presented by Chow and Kohler [Chow79] but these models are not appropriate for the analysis of loosely coupled systems [Chu80].

The work of Casey and Shelness [Case77a, Case77b] described one method for performing dynamic load balancing, but their algorithm allows jobs to be queued while awaiting resources, which seems to be unsuitable for an interactive system such as those assumed in this research since a job could be queued for a long time while the user is waiting for its completion. The authors

state that deadlock is also a possibility in their model under conditions of heavy loading.

Previous research on the placement problem seems to be less influenced by simplifying assumptions than the research on the assignment problem. The limitations of some of the earlier models have been considerably reduced in more recent work. For example, the interrelationship between programs and data was ignored by Chu [Chu69, Chu73] and Casey [Case72] but was addressed by Morgan and Levin [Morg77, Levi75] and by Fisher and Hochbaum [Fish80]. However, as pointed out by Chang and Liu [Chan79b], great care should be taken in using all of these models since cost functions are assumed to be linear while in fact, cost functions are usually non-linear. Also these mathematical programming approaches assume that costs are known and fixed, thereby ignoring variable queueing delays and the overhead associated with remote processing as opposed to local processing.

Finally, none of the research on the assignment problem appears to account for all of the complexities associated with the existence of multiple copies of data files. For example, the recent work of Chen and Akoka [Chen80] correctly differentiates between the cost of data file queries versus the cost of data file updates, as do Morgan and Levin [Morg77, Levi75], but it does not distinguish between the complexity of the query and the update transaction when a data file may have several copies located at several locations. A query transaction needs to accesss only one of the many copies of a data file whereas an update transaction needs access to all copies of the data. Furthermore, a query transaction can share a data file copy with another query, while an update usually requires exclusive access to the data file copy during the period of the update. Another important factor is that the cost of updating N copies of a data file is not equal to N times the cost of updating one copy.

SECTION 3

The previous research on the assignment of executable processes to processors in a distributed system has been limited to special cases which are not very representative of large, loosely-coupled, fully distributed processing systems. This dissertation investigates work distribution in fully distributed systems in an environment that is as realistic as possible. For this reason, as well as others, a decision was made to use a simulation model to study the work distribution problem. The use of an analytical model was considered but was not deemed practical because of the complex nature of the problem. Likewise, the use of an actual system which could test all of the variables of interest would be prohibitively expensive and time consuming to implement. Thus, a simulation model seems to be the best choice for at least the first steps in this type of research.

3.1 Description of the Simulator

The simulator developed for this purpose is event-based and is programmed in PASCAL. The simulator models a fully distributed network as a set of nodes (machines) connected by communication links over which messages will pass. Each node is modeled as a single, time-shared computer. Multiple processes can reside in each machine. All communication between processes is by means of messages. Each node (see Figure 6) contains a READY QUEUE of processes which are ready to execute, a BLOCKED QUEUE of processes awaiting a message, a MESSAGE QUEUE of messages which have just arrived at this node, and a process which is currently active at this node. PORT QUEUES are used to direct messages to the proper process within a node. Processes are activated in a time-sharing manner, but message handling has priority over other processing. All queues are of the first-in-first-out (FIFO) type with no preemption.

In the simulation model, a link is a uni-directional path between nodes over which messages may travel. Any number of links may connect two nodes. Each link contains a LINK QUEUE of messages awaiting transmission on the link and an active message which is currently being transmitted.

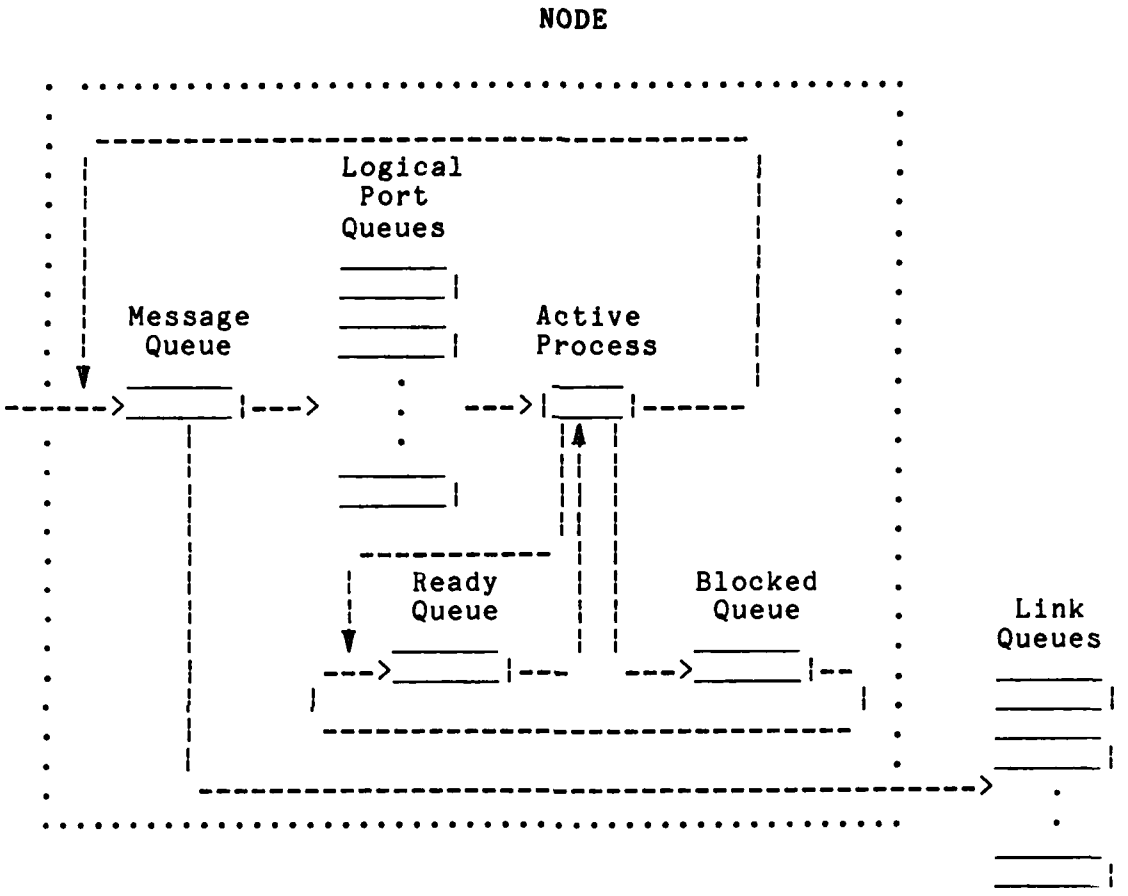


Figure 6. Models of a Node and a Link

A process in the model performs a series of actions called a script. A script can contain processing actions, send actions, receive (wait) actions, and can loop and stop. Various scripts were implemented for the simulation runs.

The simulator used in this experiment was also used for research on various models of executive control [Ensl81a, Ensl81b, and Sapo81]. The executive control provided by the simulator includes the functions of distributed and decentralized control (a function of the network operating system), functions typically provided by a local operating system, and system support functions such as file processes which are activated to provide user processes access to data files. Details of these functions can be found in [Sapo81].

5.1.5 The Simulator Design

The simulator is composed of the following modules: node module, message system module, file system module, command interpreter module, task set and process manager module, and a load generator module. The bulk of the code representing the simulated executive control is contained in the FILE SYSTEM and TASK SET AND PROCESS MANAGER modules.

5.1.5.1 Node Module

The NODE MODULE simulates the hardware activities of each node (e.g., the processor and attached disks). This includes the simulation of user activities as specified by process scripts and the simulation of disk traffic. In addition, this module provides the local operating system functions of dispatching, blocking processes for message transmission or reception, and unblocking of processes.

5.1.5.2 Message System

All activities dealing with messages are handled by the MESSAGE SYSTEM. Among the services provided by this module are the following: 1) routing of messages, 2) placement of messages in LINK QUEUEs, 3) transmission of messages across a link, 4) transmission of acknowledgements to the source end of a link, and 5) placement of messages in PORT QUEUEs.

5.1.5.3 File System

The FILE SYSTEM handles the various types of simulated files, which include object, command, and data files. It stores the scripts for object files and

provides access to the scripts. Similarly for command files, it stores the work requests for each command file and controls access to the file. It is in this module that the file management strategy for each model of control is simulated. The reader is referred to Chapter IV for a description of each control model including specific details concerning the file management strategies that are simulated.

5.1.5.4 Command Interpreter

The COMMAND INTERPRETER parses work requests and constructs the task graph describing the initial resource requirements for a work request.

5.1.5.5 Task Set and Process Manager

The TASK SET AND PROCESS MANAGER performs all control activities required to manage all phases of execution of a work request. This includes activating the COMMAND INTERPRETER; communicating with the FILE SYSTEM in order to gather information, allocate files, or deallocate files; performing work distribution and resource allocation; and managing active processes.

5.1.5.6 Load Generator

Work request traffic originating from the user terminals attached to each node is simulated by the LOAD GENERATOR. A series of work requests provided by a user at a terminal is called a user session. To simulate a user session, the LOAD GENERATOR randomly chooses a session length from an interval specified by the experimenter. A session starting time (measured in seconds) is also chosen at random from an interval specified by the experimenter. Each work request for the user session is chosen at random from the population of work requests originally created for each node via the input statements described above. The LOAD GENERATOR also simulates the "think time" between work requests by randomly choosing a time (measured in seconds) from another interval specified by the experimenter.

In the control model used for this experiment, each work request is received from a user, the work request is parsed by the TASK SET MANAGER, and a task graph is generated, file availability is determined by sending messages to the FILE SYSTEM MANAGER at the various nodes, processor utilization is determined by sending messages to the PROCESSOR UTILIZATION MANAGERs, and when resource availability information is complete, the task graph is given to the RESOURCE ALLOCATION AND WORK DISTRIBUTION MODULE to assign processes to proces-

sors for execution. After the work distribution decision is made, the process assignments and decisions to move files prior to process execution are indicated in the task graph which is returned to the executive control for execution of the work request. If a work distribution decision can not be made because all needed resources are not available, then an error indicator is returned to the executive control.

Execution of a work request is performed by sending messages to FILE PROCESSES to copy files from node to node, if required, and by sending messages to PROCESS MANAGERS at each node to initiate execution of the appropriate processes. When a process terminates, a message is sent to the TASK SET MANAGER. When all processes for a work request have terminated, the user is informed and a new work request can be accepted from that user.

3.2 Input to the Simulator

Input to the simulator consists of three parts, the work requests, network configuration information, and information about the file system. The network configuration information contains data about the physical configuration of the simulated network. The network configuration information contains node and link information. All node information remained constant during this experiment. The values for node information used in this experiment were as follows: (1) processing speed (1,000,000 instructions per second), (2) length of a time slice (300 microseconds), (3) number of attached user terminals per processor (10), (4) number of attached disks per processor (3), (5) disk transfer speed (500,000 bytes per second), and (6) average disk latency (100 microseconds). Link information includes the source and destination of each half-duplex communication link and the bandwidth of the link. The number of links and their source and destination were varied during the experiment to simulate different network configurations. The network configurations which were simulated were a fully-connected network, a uni-directional ring, and a global bus. The network configurations will be explained more fully in the discussion of the experimental procedure.

The work requests define the load placed on the system by users attached to the system by means of interactive terminals. The simulator was designed to accept work requests generated from command files as well as the spawning of new processes during the execution of a process. However, in this

experiment only user work requests originating from terminals were simulated. The simulator generated work requests for each of the ten users at each of the five nodes of the network by selecting a work request at random from a pool of work requests which were input into the simulator. After the completion of each work request, a new work request was generated for that user after a time delay which varied randomly from one to two seconds. The pool of work requests which were available for selection contained only one type of work request for each test case, but the type of work request was varied for the different test cases. The specific types of work requests which were tested are explained in the discussion of the experimental procedure.

Information about the location and availability of data files and object files is collected by the executive control by contacting each node prior to invocation of the work distribution algorithm. This information about file availability is made available to the work distribution algorithm each time it is invoked. If a file exists at more than one node, then each location of that file is noted. Data files are locked while they are being written and cannot be accessed by other processes during that period. Data files and/or object files may be copied from one node to another based on the work distribution decision. In this case, the new copy of the file is temporary and is destroyed after the work request which uses it is completed.

The size of each data file and object file is used by the simulator, but no actual data is present. Object files contain a file size and a script for executing the process. The script can represent any sequence of the following five actions: (1) compute a certain number of instructions, (2) send a message, (3) receive a message, (4) loop back to a previous command a given number of times, and (5) terminate the process. Details of the scripts that were actually used during the simulation are presented in the discussion of the experimental procedure.

The simulator considers the time required for system overhead, including the time to execute each work distribution algorithm, and includes this overhead time in the time to process a work request. The complexity of each of the three work distribution algorithms is on the order of $L^2 * N^2$ where L is the number of logical components in the work request and N is the number of physical nodes in the network (N = 5 in this experiment). As a comparison,

the complexity of an algorithm which considers allpossible assignments of logical components (L) to physical nodes (N) is on the order of $L! * N^L$.

3.3 The Work Distribution Algorithms

Three different work distribution algorithms were tested in the simulation experiments. The first work distribution algorithm attempts to minimize communications between nodes of the network. A second algorithm attempts to balance the workload on the various system processors. A third algorithm combines the goals of the other two algorithms. Average user response time is the performance measure which all three of these algorithms try to minimize.

The first work distribution algorithm (minimize data communication) is described in Figure 7. To simplify the presentation of this algorithm, only single copies of files are considered in this explanation. The complete algorithm is given in Appendix A. It can handle multiple file copies as well as single copies. This algorithm tries to minimize communication by trading a file copy and transfer operation for the remote accessing of data across the network. The "minimize communication" algorithm considers each process in the task graph in the order in which it is parsed by the Command Interpreter (i.e., each object file in order from left to right in the command line followed by each data file from left to right in the command line. If the process is already assigned, then the algorithm continues to the next process.

For each process being considered for assignment, the communication with other processes in the task graph is considered. The total time for this process to communicate with each other process is computed, assuming that all processes remain in their initial locations. The communication time is calculated by dividing the message traffic between each pair of nodes by the effective message speed between the two nodes. The effective message speed for each communication link traversed is calculated by dividing the rated bandwidth of the link by the number of messages in the link queue (including the present message) waiting to be sent on that link. Next, the same calculations are performed, assuming that the process being considered for assignment is copied to the location of each process linked to it in the task graph. In this case, the time to copy the file (code) for the process is added to the time to communicate with the other processes from the new location. The file copy time is calculated by dividing the size of the file to be copied by the

1. Consider each process in the task graph in the order in which it was parsed by the Command Interpreter.
2. If the process is already assigned, then get the next process.
3. For this process, consider all processes linked to it in the task graph.
4. Compute the total communication time between this process and all other processes linked to it in the task graph.
5. Consider copying this process to the location of each linked process and compute the total communication time (including copying time) between this process (at its new location) and all other processes linked to it in the task graph.
6. Compare each communication time in step 5 with the communication time calculated in step 4 and choose the minimum.
7. If one of the locations in step 5 was chosen, then plan to copy the process to the chosen location (and update the location/availability portion of the task graph) and assign this process to the location chosen (and update the "Copy from Node" and the "Assign to Node" entries of the task graph).
8. If the original location in step 4 was chosen, then assign the process to its current location (and update the "Assign to Node" entry in the task graph).
9. Continue until all processes in the task graph have been assigned.

Figure 7. Minimize Communication Algorithm

effective message speed. The minimum total time is selected.

If the minimum total time corresponded to the initial location of the process, then the process is assigned to that node. The "Assign to Node" entry for that process in the task graph is updated to reflect this assignment. If the minimum time corresponds to a new location for the process, then the "Copy from Node" entry in the task graph is updated with the initial location of the process and the "Assign to Node" entry is updated with the selected location. The location/availability portion of the task graph also is updated to reflect the new location of the process.

Each process in the task graph is assigned in turn. After all processes in the task graph have been assigned, the task graph will indicate the node selected for each process and information (if any) about copying necessary to move files from node to node prior to execution.

The second work distribution algorithm (load balancing) is outlined in Figure 8. Again for simplicity of presentation, the description in Figure 8 considers only single copies of files whereas multiple copies are handled by the complete algorithm presented in Appendix B. The "load balancing" algorithm also considers each process in the task graph in order, and, if that process is already assigned, it goes on to the next.

For each process being considered for assignment, its processing time on each system processor (node) is considered. The processing workload for the process is obtained from the task graph also. The total processing time at the node where the process is initially located is calculated first. The processing time is calculated by dividing the number of instructions in the process by the effective processor speed. The effective processor speed is calculated by dividing the rated processor speed (1 million instructions/second) by the number of processes (counting the present process) which are currently executing on that processor. This time is compared to the total of the processing time at each other processor (node) in the network plus the time needed to copy the process from its initial location to the node under consideration. The minimum total time is selected. If the minimum total time corresponds to the initial location of the process, then the process is assigned to that node and the task graph is updated as before. If a decision is made to copy

1. Consider each process in the task graph in the order in which it was parsed by the Command Interpreter.
2. If this process is already assigned, then go to the next process.
3. Calculate the time required to execute the process at its present location (based upon the estimated processor speed).
4. Calculate the time required to move the executable file (code) for that process to each other node (processor) in the system plus the time to execute the process at the new node. Choose the location with the minimum total time.
5. If the minimum total time in step 4 is less than the time calculated in step 3, then choose the location from step 4.
6. If another location is chosen, then plan to copy the process to the chosen location (and update the location/availability portion of the task graph) and assign this process to the location chosen (and update the "Copy from Node" and the "Assign to Node" entries in the task graph).
7. If the original location was chosen, then assign the process to its current location (and update the "Assign to Node" entry in the task graph).

Figure 8. Load Balancing Algorithm

the process to another node prior to execution, then that information is reflected in the updated task graph. Each process in the task graph is assigned in this manner until all processes have been assigned.

The third work distribution algorithm (combination) attempts to minimize the total of communication time and processing time. The "combination" algorithm is shown in Figure 9. Again, only single file copies are considered, and the complete algorithm is presented in Appendix C. Whereas the first algorithm considered only processors (nodes) where linked processes already existed, the "combination" algorithm (like the load balancing algorithm) considers all processors (nodes) as candidates for assignment.

The "combination" algorithm, like the other two, considers each process in the task graph in order, and, if that process is already assigned, goes on to the next. For each process being considered for assignment, the time to execute the process at its current location is calculated and is added to the time to communicate with all processes linked to it in the task graph. The sum of the processing time and the communication time produces an estimate of the total execution time. Next, the process under consideration is assumed to be copied to every other processor (node) in the network. For each other assumed location of the process, the total of processing time, communication time, and copying time is calculated. The location with the minimum total time is the one selected for assignment.

If the minimum total time corresponds to the initial location of the process, then the process is assigned to that node and the task graph is updated as before. If a decision is made to copy the process to another node prior to execution, then that information is used to update the task graph. Each process in the task graph is assigned an execution location in this manner until all processes have been assigned.

3.4 Procedure

In order to compare the performance of the three work distribution algorithms in various environments, two sets of experiments were performed to measure the average user response time and the system throughput while using

1. Consider each process in the task graph in the order in which it was parsed by the Command Interpreter.
2. If the process is already assigned, then get the next process.
3. Calculate the time required to execute this process at its present location.
4. For this process, consider all processes linked to it in the task graph.
5. Compute the communication time between this process and all other processes linked to it in the task graph. Add the processing time (from step 3) to the communication time calculated in this step to obtain an estimate of total executing time.
6. Now consider copying this process to each other node in the system. Calculate the read time to copy the file (code) for the process to each new location. Calculate the processing time at the new location.
7. For each other location, consider the communications with all other processes linked to this process in the task graph.
8. Compute the communication time between this process and all other processes linked to it in the task graph. Add the processing time and the copying time (from step 6) to the communication time calculated in this step to get the total time.
9. Select the minimum total time from step 5 and step 8.
10. If one of the locations in step 8 was chosen, then plan to copy the process to the chosen location (and update the location/availability portion of the task graph) and assign this process to the location chosen (and update the "Copy from Node" and the "Assign to Node" entries of the task graph).
11. If the minimum total time from step 5 was chosen, then assign the process to its current location (and update the "Assign to Node" entry of the task graph).
12. Continue until all processes in the task graph have been assigned.

Figure 9. Combination Algorithm

each of the three work distribution algorithms under different test conditions.

3.4.1 Experiments with Different Types of Work Requests

In the first set of experiments, the performance of the work distribution algorithms were compared in test cases which examined all possible combinations of: (1) three types of work requests, (2) three network topologies, and (3) two conditions of file redundancy.

The first type of work request consists of a process which reads input from one data file and writes output to another data file. This work request was presented to the simulator as the command:

```
i> p >o
```

where i represents the input data file, p represents the process, and o represents the output data file. The > sign directs input to the process and output from the process. A pool of work requests was input to the simulator. Each of these had a unique file name, and each file name had a single, unique location. The location of the input file (i), the object file (p), and the output file (o) varied randomly depending on the specific work request that was selected. The script for the main process (p) for this type of work request is as follows:

```
r  1          (read a message from port 1)
c 10000       (simulate 10,000 instructions)
s  2  1024    (send a message to port 2)
l  1  29      (loop back to the beginning 29 times)
t             (terminate)
```

Since all data messages in the simulator are 1024 bytes long, this work requests reads 30 K bytes of data from the input file (i) and writes 30 K bytes of data to the output file (o).

The second type of work request used in this experiment is similar to the first one but has two input files and two output files. These work requests are presented to the simulator by the command:

```
i1> i2> p >o1 >o2
```

where i1 and i2 represent input data files and o1 and o2 represent output data files. The initial location of each of the files is again chosen randomly. The script for this process (p) is as follows:

```
r  1          (read a message from port 1)
c  10000      (simulate 10,000 instructions)
```

```
s  2  1024  (send a message to port 2)
r  2        (read a message from port 2)
s  1  1024  (send a messages to port 1)
l  1  29    (loop back to the beginning 29 times)
t           (terminate)
```

The third group of work requests requires that data be piped between three processes. This type of work request is represented in the simulator by the command:

```
i> p1 | p2 | p3 >o
```

where i is a single input data file, o is a single output data file, and p1, p2, and p3 are three processes. Process p1 reads data from the input file (i) and pipes it to process p2 which pipes it to process p3 which writes data to the output file (o). The scripts for each of the processes (p1, p2, and p3) is the same and is as follows:

```
r  1        (read a message from port 1)
c  10000    (simulate 10,000 instructions)
s  1  1024  (send a message to port 1)
l  1  29    (loop back to the beginning 29 times)
t           (terminate)
```

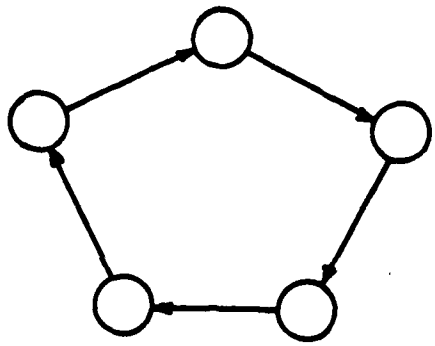
Thus, 30 K bytes of data are read from the input file (i) by p1, are piped to p2, are piped to p3, and are written to the output file (o).

The three network topologies simulated were a uni-directional ring, a fully-connected network, and a global bus. Figure 10 illustrates each of these.

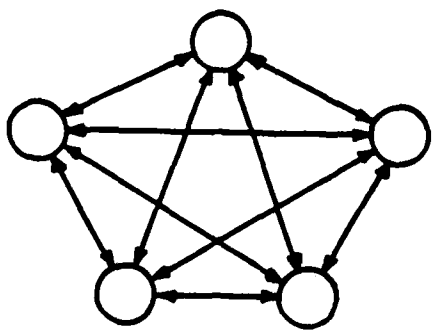
The fully-connected network has two links connecting each node to each other node in the network, one in each direction. Thus there are 20 half-duplex links in this network configurations simulating 10 full-duplex connections. The longest path from one node to another node is one link. The uni-directional ring has 5 half-duplex links. Each node has a single predecessor node and a single successor node. The longest path from one node to another node is four hops. In the global bus configuration, there is only one link for the entire network and it is shared by all nodes on a round-robin basis.

The two kinds of file redundancy which were simulated in this experiment are the single file case and the multiple file case. In the first case, each data file and each object file had only one copy, randomly assigned to one of the nodes of the network. In the second case, each file had three copies, with their locations again randomly selected.

A UNI-DIRECTIONAL RING TOPOLOGY



A FULL-CONNECTED NETWORK



A GLOBAL BUS TOPOLOGY

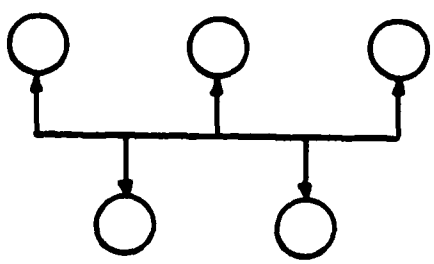


Figure 10. Examples of Three Network Topologies

Combining each of the network configurations with each type of work request and each type of file redundancy produced 18 test cases. Each of the test cases was executed with each of the three work distribution algorithms and results were produced for each. In addition, each case was run three times with each algorithm to obtain enough data to perform a statistical analysis of the results.

3.4.2 Experiments with Different Bandwidths

In the second set of experiments, the performance of the work distribution algorithms was compared in test cases which examined all possible combinations of: (1) four different communication bandwidths, (2) three network topologies, and (3) two conditions of file redundancy. The three network topologies and the two conditions of file redundancy are the same ones that were tested in the first set of experiments. The four bandwidths which were tested in this set of experiments are as follows:

50,000 bytes/second
100,000 bytes/second
500,000 bytes/second
2,500,000 bytes/second.

The workload for this set of experiments consisted of a mixture of 40% Type 1 Work Requests (i> p >o), 40% Type 2 Work Requests (i1> i2> p >o1 >o2), and 20% Type 3 Work Requests (i> p1 | p2 | p3 >o). This mixture of work requests was a controlled mixture in the sense that after a Type 1 Work Request was completed, another Type 1 Work Request was initiated (after a user "think time" interval). An earlier strategy which randomly selected the next work request to start from a pool of 40% Type 1 Work Requests, 40% Type 2 Work Requests, and 20% Type 3 Work Requests was abandoned because longer-running work requests stayed in the system while shorter work requests were gradually replaced by more and more longer jobs, so that the system was very slow to reach the steady-state mix of jobs which would be heavily weighted by long-running work requests. The controlled mixture guarantees that the mix of work requests in the system will remain fixed over time.

3.5.1 Outputs from the Simulator

The output from each run of the simulator consists of an assignment trace and a listing of various statistics for that run. The assignment trace contains a record of the assignments that were made for each work request. It contains the time the assignment was made and the initial location of each

file as well as the selected location for execution of each component of the work request. If a process is to be moved prior to execution, it shows the from-location and the to-location for the move. A sample assignment trace is shown in Appendix E.

The listing of statistical results for each run contains the following items: (1) the time duration of the simulation, (2) the inputs to the simulation run, (3) the average response time for work requests at each node, (4) the average response time for processes at each node, (5) the mean length of the ready queue at each node, (6) the mean length of the blocked queue at each node, (7) the mean length of each disk queue, (8) the mean length of each link queue, (9) the message traffic on each link, and (10) the specific work requests selected from the pool at each node.

SECTION 4

The simulation experiment consisted of two parts. In the first set of experiments, the communication bandwidth was held constant and performance measures of average user response time and system throughput were obtained for each of the three work distribution algorithms with workloads consisting of only one type of work request at a time. The system topology was varied in this set of experiments and the effects of multiple copies of files as well as single copies of files were also examined. This part of the experiment served to validate the simulator and verify that the assignments made by the work distribution algorithm were correct. It also provided information about the relative performance of the different algorithms with different types of jobs.

The second set of experiments showed the relative performance of the work distribution algorithms at different communication bandwidths. A controlled mixture of the three different types of work requests used in the first set of experiments was used as the workload in the second set of experiments. As before, performance of the three work distribution algorithms was compared for three different network topologies and for single and multiple file copies.

4.1 Experiments with Different Types of Work Requests

The mean and standard deviation of the average user response time for the first set of experiments are presented in Tables 1-3. Performance was measured for each simulation run by calculating the average user response time for all work requests which were executed during the specified time interval. Between 35 and 150 work requests were completed in each run. Each test case was run three times with each work distribution algorithm. Tables 4-6 show the mean and standard deviation of system throughput values corresponding to the average user response time values in Tables 1-3. The individual values of average user response time and system throughput for each of the three runs for each experimental condition. from which the mean and standard deviation are calculated, are shown in Appendix F.

Table 1. Means and Standard Deviations of
Average User Response Time for a Uni-Directional Ring

SINGLE FILE COPIES

Case No.	Work Requests	Algorithm 1	Algorithm 2	Algorithm 3	Base-Line Algorithm
1	Type 1	2.9140	3.0802	2.7408	2.7675
		0.1457	0.1026	0.2860	0.1010
2	Type 2	4.4566	4.8078	5.2305	4.8985
		0.1421	0.2516	0.3593	0.1928
3	Type 3	6.4213	5.6532	6.0135	5.4427
		1.3426	0.9497	0.7860	0.4325

MULTIPLE FILE COPIES

Case No.	Work Requests	Algorithm 1	Algorithm 2	Algorithm 3
4	Type 1	2.4527	2.9509	2.7223
		0.0593	0.0825	0.1431
5	Type 2	3.1396	4.5940	3.3049
		0.0578	0.1070	0.1708
6	Type 3	6.1737	4.8546	4.3239
		1.5420	0.7071	0.7851

Note: The top line is the mean value.
The second line is the standard deviation.

Table 2. Means and Standard Deviations of
Average User Response Time for a Fully-Connected Network

SINGLE FILE COPIES

Case No.	Work Requests	Algorithm 1	Algorithm 2	Algorithm 3	Base-Line Algorithm
7	Type 1	2.3157 0.2590	2.5869 0.1328	2.3624 0.0671	2.3632 0.1721
8	Type 2	2.7182 0.0553	2.8854 0.1025	3.0705 0.1016	2.6711 0.0419
9	Type 3	4.4174 0.5991	4.6929 0.4991	4.1563 0.3767	4.0040 0.9408

MULTIPLE FILE COPIES

Case No.	Work Requests	Algorithm 1	Algorithm 2	Algorithm 3
10	Type 1	2.1366 *0.0758*	2.6123 *0.3497*	2.4359 *0.0679*
11	Type 2	2.6739 0.0191	2.8285 0.1042	2.6861 0.0711
12	Type 3	4.1078 0.1581	3.9381 0.5350	3.9455 0.3828

Note: The top line is the mean value.
The second line is the standard deviation.

Table 3. Means and Standard Deviations of
Average User Response Time for a Global Bus Network

SINGLE FILE COPIES

Case No.	Work Requests	Algorithm 1	Algorithm 2	Algorithm 3	Base-Line Algorithm
13	Type 1	3.8405	4.8196	3.9222	4.4505
		0.1442	0.3434	0.1244	0.0183
14	Type 2	7.7860	9.5666	8.1923	9.6272
		0.1353	0.2702	0.4937	0.1547
15	Type 3	5.4268	6.5294	5.5233	7.5363
		0.2553	1.7237	0.4023	2.0617

MULTIPLE FILE COPIES

Case No.	Work Requests	Algorithm 1	Algorithm 2	Algorithm 3
16	Type 1	2.1962	4.1079	2.3136
		0.1326	0.2341	0.1180
17	Type 2	4.2159	5.7520	3.1589
		0.2083	0.4247	0.0594
18	Type 3	5.3354	7.2135	5.4262
		0.5129	0.1389	0.2954

Note: The top line is the mean value.
The second line is the standard deviation.

Table 4. Means and Standard Deviations of System Throughput for a Uni-Directional Ring

SINGLE FILE COPIES

Case No.	Work Requests	Algorithm 1	Algorithm 2	Algorithm 3	Base-Line Algorithm
1	Type 1	11.4	10.5	11.7	12.0
		0.5	0.8	1.8	0.4
2	Type 2	9.0	8.0	3.8	8.2
		0.5	0.3	0.7	0.3
3	Type 3	3.5	4.9	4.0	5.0
		0.6	0.5	0.5	0.3

MULTIPLE FILE COPIES

Case No.	Work Requests	Algorithm 1	Algorithm 2	Algorithm 3
4	Type 1	12.7	11.8	11.2
		0.2	0.1	0.3
5	Type 2	10.6	8.7	10.3
		0.4	0.3	0.6
6	Type 3	4.8	4.7	4.8
		0.4	0.4	0.5

Note: The top line is the mean value.
The second line is the standard deviation.

Table 5. Means and Standard Deviations of System Throughput for a Fully-Connected Network

SINGLE FILE COPIES

Case No.	Work Requests	Algorithm 1	Algorithm 2	Algorithm 3	Base-Line Algorithm
7	Type 1	12.1	10.6	12.4	13.2
		0.5	0.3	1.2	0.9
8	Type 2	11.5	10.7	11.1	11.7
		0.7	0.2	0.4	0.3
9	Type 3	5.0	5.2	2.7	5.3
		0.4	0.5	1.1	0.5

MULTIPLE FILE COPIES

Case No.	Work Requests	Algorithm 1	Algorithm 2	Algorithm 3
10	Type 1	13.2	11.5	12.9
		0.3	0.9	0.6
11	Type 2	12.3	10.7	11.8
		0.4	0.2	0.6
12	Type 3	5.4	4.5	4.9
		0.3	0.6	0.3

Note: The top line is the mean value.
The second line is the standard deviation.

Table 6. Means and Standard Deviations of System Throughput for a Global Bus Network

SINGLE FILE COPIES

Case No.	Work Requests	Algorithm 1	Algorithm 2	Algorithm 3	Base-Line Algorithm
13	Type 1	9.6	7.7	7.3	8.7
		0.5	0.6	0.5	0.5
14	Type 2	5.0	4.7	5.2	4.2
		0.3	0.3	0.4	0.4
15	Type 3	4.5	4.1	4.4	4.0
		0.4	0.2	0.2	0.2

MULTIPLE FILE COPIES

Case No.	Work Requests	Algorithm 1	Algorithm 2	Algorithm 3
16	Type 1	13.3	9.5	11.2
		0.3	1.0	0.5
17	Type 2	8.2	8.5	10.1
		1.3	0.5	0.6
18	Type 3	4.4	4.1	4.5
		0.2	0.3	0.2

Note: The top line is the mean value.
The second line is the standard deviation.

User response time is defined as the difference between the starting time of a work request and its completion time. Averge user response time is calculated by averaging the user response time for all work requests which complete in the time interval being measured. System throughput is simply the number of work requests completed in one second for the entire system. It is computed by dividing the number of work requests which were completed in the interval being measured by the length of the interval.

For those test cases with single file copies, the results of running a Base-Line Algorithm also are presented. The Base-Line Algorithm executes every process where it is initially located, so that no files are moved. The inclusion of this algorithm allows comparisons of Algorithms 1, 2, and 3 with a passive or "do nothing" assignment policy which can be considered as a standard or base-line of performance. Since the Base-Line Algorithm can not handle multiple file copies, comparisons with the Base-Line Algorithm are not appropriate for test cases with multiple file copies. A complete listing of the Base-Line Algorithm is presented in Appendix D.

The results which are presented in Appendix F were analyzed using Student's t-test to determine if differences in average response time for two algorithms in a test case were significant. Student's t-test is used for determining differences between the means of small samples [Freu79]. Tables 7, 8, and 9 show the results of t-test calculations between each pair of algorithms for each test condition. Table 10 shows the results of t-test calculations between test cases where the only difference is single or multiple copies of files. Table 10 also shows t-test differences between each run with multiple file copies and the corresponding test case with single file copies and the Base-Line Algorithm. For example, with Type 1 Work Requests and a uni-directional ring topology, the results for Algorithm 1 with multiple copies (Test Case 4) is compared against Algorithm 1 with single copies (Test Case 1) and also against the Base-Line Algorithm for Test Case 1.

The t-test values are computed using the following equation:

$$t(1,2) = \frac{mean(1) + mean(2)}{\sqrt{\frac{sd(1) \times sd(1) + sd(2) \times sd(2)}{3}}}$$

Table 7. Values of t for the Different Work Distribution Algorithms with a Uni-Directional Ring.

EXPERIMENTS WITH DIFFERENT TYPES OF WORK REQUESTS

Case No.	Work Requests	File Copies	t-values t(1,2)	t(1,3)	t(2,3)	t(1,0)	t(2,0)	t(3,0)
			Average User Response Time					
1	Type 1	Single	1.6	0.9	1.9	1.4	3.8*	1.5
2	Type 2	Single	2.1	3.5*	1.7	3.2*	0.5	1.4
3	Type 3	Single	0.8	0.7	0.3	1.2	0.3	0.8
4	Type 1	Multiple	8.5*	3.0*	2.4			
5	Type 2	Multiple	20.7*	1.6	11.1*			
6	Type 3	Multiple	1.3	1.9	0.9			
			System Throughput					
1	Type 1	Single	1.7	0.3	1.1	1.6	2.9*	0.3
2	Type 2	Single	3.0*	10.5*	9.6*	2.4	0.8	10.0*
3	Type 3	Single	3.1*	1.1	2.2	3.9*	0.3	3.0*
4	Type 1	Multiple	7.0*	7.2*	3.3*			
5	Type 2	Multiple	6.6*	0.7	4.1*			
6	Type 3	Multiple	0.3	0.0	0.3			

* Significant at the 95% level (t>2.7).

Note: The t-value t(1,2) compares the performance of Algorithms 1 and 2.

Table 8. Values of t for the Different Work Distribution Algorithms with a Fully-Connected Network

EXPERIMENTS WITH DIFFERENT TYPES OF WORK REQUESTS

Case No.	Work Requests	File Copies	T-values t(1.2)	t(1,3)	t(2,3)	t(1,0)	t(2,0)	t(3,0)
			Average User Response Time					
7	Type 1	Single	1.6	3.0*	2.6	0.3	1.8	0.0
8	Type 2	Single	2.5	5.3*	2.2	1.2	3.4*	6.3*
9	Type 3	Single	0.6	0.6	1.5	0.6	1.1	0.3
10	Type 1	Multiple	2.3	5.1*	0.9			
11	Type 2	Multiple	2.5	0.3	2.0			
12	Type 3	Multiple	0.5	0.7	0.0			
			System Throughput					
7	Type 1	Single	4.5*	0.4	2.5	1.9	4.7*	0.9
8	Type 2	Single	1.9	0.9	1.5	0.5	4.8*	2.1
9	Type 3	Single	0.5	3.4*	3.6*	0.8	0.2	3.7*
10	Type 1	Multiple	3.1*	0.8	2.2			
11	Type 2	Multiple	6.2*	1.2	3.0*			
12	Type 3	Multiple	2.3	2.0	1.0			

* Significant at the 95% level (t>2.7).

Note: The t-value t(1,2) compares the performance of Algorithms 1 and 2.

Table 9. Values of t for the Different Work Distribution Algorithms with a Global Bus Network

EXPERIMENTS WITH DIFFERENT TYPES OF WORK REQUESTS

Case No.	Work Requests	File Copies	T-values t(1,2)	t(1,3)	t(2,3)	t(1,0)	t(2,0)	t(3,0)
			Average User Response Time					
13	Type 1	Single	4.6*	0.7	4.3*	7.3*	1.9	7.3*
14	Type 2	Single	10.2*	1.4	4.2*	15.5*	0.3	4.8*
15	Type 3	Single	1.1	0.4	1.0	1.8	0.6	1.7
16	Type 1	Multiple	12.3*	1.1	11.9*			
17	Type 2	Multiple	5.6*	8.5*	10.5*			
18	Type 3	Multiple	6.2*	0.3	9.5*			
			System Throughput					
13	Type 1	Single	4.2*	5.6*	0.9	2.2	2.2	3.4*
14	Type 2	Single	1.2	0.7	1.7	2.8*	1.7	3.1*
15	Type 3	Single	1.5	0.4	1.8	1.9	0.6	2.4
16	Type 1	Multiple	6.3*	6.2*	2.6			
17	Type 2	Multiple	0.4	2.3	3.5*			
18	Type 3	Multiple	1.4	0.6	1.9			

* Significant at the 95% level (t>2.7).

Note: The t-value t(1,2) compares the performance of Algorithms 1 and 2.

Table 10. Values of t Comparing Single and Multiple File Copies

EXPERIMENTS WITH DIFFERENT TYPES OF WORK REQUESTS

Work Requests	T-values t(1,1)	t(2,2)	t(3,3)	t(1,0)	t(2,0)	t(3,0)
Average User Response Time						
Uni-Directional Ring						
Type 1	5.1*	1.7	0.1	4.7*	2.4	0.4
Type 2	14.9*	1.4	8.4*	15.1*	2.4	10.7*
Type 3	0.2	1.1	2.4	0.8	1.2	2.2
Fully-Connected Network						
Type 1	1.2	0.1	1.3	2.1	1.1	0.7
Type 2	1.3	0.7	5.4*	0.1	2.4	0.3
Type 3	0.9	1.8	0.7	0.2	0.1	0.1
Global Bus Network						
Type 1	14.5*	3.0*	16.3*	29.2*	2.5	31.0*
Type 2	24.9*	13.1*	17.5*	36.1*	14.9*	67.6*
Type 3	0.3	0.7	0.3	1.8	0.3	1.8
System Throughput						
Uni-Directional Ring						
Type 1	4.2*	2.8*	0.5	2.7*	0.8	2.8*
Type 2	2.7*	2.9*	52.0*	8.3*	2.0	4.5*
Type 3	3.1*	0.5	2.0	0.7	1.0	0.6
Fully-Connected Network						
Type 1	3.3*	1.6	0.6	0.0	2.3	0.5
Type 2	1.7	0.0	1.7	2.1	4.8*	0.3
Type 3	1.4	1.6	3.3*	0.3	1.8	1.2
Global Bus Network						
Type 1	11.0*	2.7*	9.6*	13.7*	1.2	6.1*
Type 2	4.2*	11.3*	11.8*	5.1*	11.6*	14.2*
Type 3	0.4	0.0	0.6	2.4	0.5	3.1*

* Significant at the 95% level (t>2.7).

Note: The t-value t(1,2) compares the performance of Algorithms 1 and 2.

where t(1,2) is the t-test value comparing the mean for Algorithm 1 with the mean for Algorithm 2, mean(1) is the mean for Algorithm 1, mean(2) is the mean for Algorithm 2, sd(1) is the standard deviation for Algorithm 1, and sd(2) is the standard deviation for Algorithm 2. The results of the t-test calculation show a significant difference with a 95% confidence interval if the t-test value is greater than 2.7. A 95% confidence level is used in the analysis of all t-test results in this dissertation. At the 99% level of confidence, the t-test value must be greater than 4.6 and must be greater than 2.1 at the 90% confidence level.

The t-test results for the first set of experiments is summarized in Table 11. This table was constructed by ordering the mean values for each algorithm in an ascending sequence for average user response time and in a descending sequence for system throughput. Then significant differences between each pair of values were indicated from the t-test results presented in Tables 7-9. Thus, the lowest (best) average user response time appears on the left and the highest (best) system throughput appears on the left also. The notation x < y in this chart means that the mean value for Algorithm x is significantly less than the mean value for Algorithm y. The notation x = y means that there is no significant statistical difference between the mean value for x and the mean value for y.

The assignment actions of the work distribution algorithms for each type of work request were observed by examining the assignment trace for each run. A sample assignment trace is shown in Appendix D. A summary of the most common assignment actions for each test condition for the first set of experiments is presented in Table 12. The assignments presented in this table were verified by pencil and paper calculation to show that the work distribution algorithms were performing the actions which they were supposed to perform. The following notation is used in Table 12:

p	a single process (object file)
p1	the first process in a pipe of three processes
p2	the second process in a pipe of three processes
p3	the third process in a pipe of three processes
i	a single input file (data file)
i1	the first of two input files
i2	the second of two input files
o	a single output file (data file)
o1	the first of two output files
o2	the second of two output files
p->i	move the object file to the location of the input file
p->best	move the object file to the least utilized processor

Table 11. Relative Rankings of the Work Distribution Algorithms

EXPERIMENTS WITH DIFFERENT TYPES OF WORK REQUESTS

Case No.	Work Requests	File Copies	Average User Response Time	System Throughput
Uni-Directional Ring				
1	Type 1	Single	3=0=1=2 (0<2)	0=3=1=2 (0>2)
2	Type 2	Single	1=2=0=3 (1<0,1<3)	1=0=2>3
3	Type 3	Single	0=2=3=1	0=2>3=1
4	Type 1	Multiple	1<3=2	1>2>3
5	Type 2	Multiple	1=3<2	1=3>2
6	Type 3	Multiple	3=2=1	1=3=2
Fully-Connected Network				
7	Type 1	Single	1<3=0=2	0=3=1>2 (3=2)
8	Type 2	Single	0=1=2=3 (1<3,0<2,0<3)	0=1=3=2 (0>2)
9	Type 3	Single	0=3=1=2	0=2=1>3
10	Type 1	Multiple	1<3=2 (1=2)	1=3=2
11	Type 2	Multiple	1=3=2	1=3>2
12	Type 3	Multiple	2=3=1	1=3=2
Global Bus Network				
13	Type 1	Single	1=3<0=2	1=0=2=3 (1>2,1>3,0>3)
14	Type 2	Single	1=3<2=0	3=1=2=0 (3>0)
15	Type 3	Single	1=3=2=0	1=3=2=0 (3>0)
16	Type 1	Multiple	1=3<2	1>3=2
17	Type 2	Multiple	3<1<2	3=2=1
18	Type 3	Multiple	1=3<2	3=1=2

Table 12. Assignment Actions for Each Test Condition

Work Request	Algorithm 1	Algorithm 2	Algorithm 3
		SINGLE FILE COPIES	
Type 1 (i> p >o)	p->i	none p->best	p->i p->o p->best
Type 2 (i1> i2> p >o1 >o2)	p->i1	none p->best	p->i p->best
Type 3 (i> p | p2 | p3 >o)	p1->p2 p3->p2	none p1->best p2->best	p1->p2 p3->p2 p3->o p1->i p1->best p2->best p3->best
		MULTIPLE FILE COPIES	
Type 1 (i> p >o)	none p->i	none p->best	none p->i p->best
Type 2 (i1> i2> p >o1 >o2)	none p->i1 p->i2 p->o1 p->o2	none p->best	none p->i1 p->i2 p->o1 p->o2 p->best
Type 3 (i> p | p2 | p3 >o)	none p1->p2 p3->p2 p2->p1 p3->p1	none p1->best p2->best p3->best	none p1->p2 p3->p2 p2->p1 p3->p1 p1->i p3->o p1->best p2->best p3->best

4.2 Experiments with Different Bandwidths

The values of average user response time for each experimental run in the second set of experiments are presented in Tables 13-18, along with the mean and standard deviation for each set of three runs for each test case. Tables 19-24 show the same type of information for system throughput for each test case. The t-test calculations were performed to compare the statistical significance of differences between the values for each pair of work distribution algorithms for each test case. The t-test results are shown in Tables 25-27 and are summarized in Table 28.

The information from Tables 13-28 is presented graphically in Figures 11-22. In each of these figures either average user response time or system throughput is plotted against bandwidth for each algorithm. The Base-Line Algorithm is designated as Algorithm 0. These graphs should be considered as histograms for each selected bandwidth rather than as smooth curves, because the bandwidth values are not placed on any kind of an interval scale. In addition to the mean values which are plotted at each bandwidth, there is also an indication of the *significant difference* between the values for each algorithm taken from Table 28.

Table 29 shows the t-test calculations between test cases where the only difference is single or multiple file copies. This table shows t-test differences between a run with multiple copies and the results with single copies using the same algorithm and using the Base-Line Algorithm.

Tables 30-35 show examples of the breakout by type of work request of the average user response time and the system throughput for the mix of jobs in Tables 13-24. The information in Tables 30-35 represents a single run for each test case. Tables 33, 34, and 35 also show a "Power Factor" for each test case which was computed by adding the throughput value for Type 1 Work Requests to two times the throughput value for Type 2 Work Requests plus three times the throughput value for Type 3 Work Requests. This "Power Factor" was constructed because the average throughput values in Tables 19-24 considered the completion of a Type 1 Work Request equal to the completion of a Type 3 Work Request even though the Type 3 Work Request performed three times as many computations as a Type 1 Work Request. The "weighting" of two given to the Type 2 Work Request in the calculation of the "Power Factor" was an arbitrary

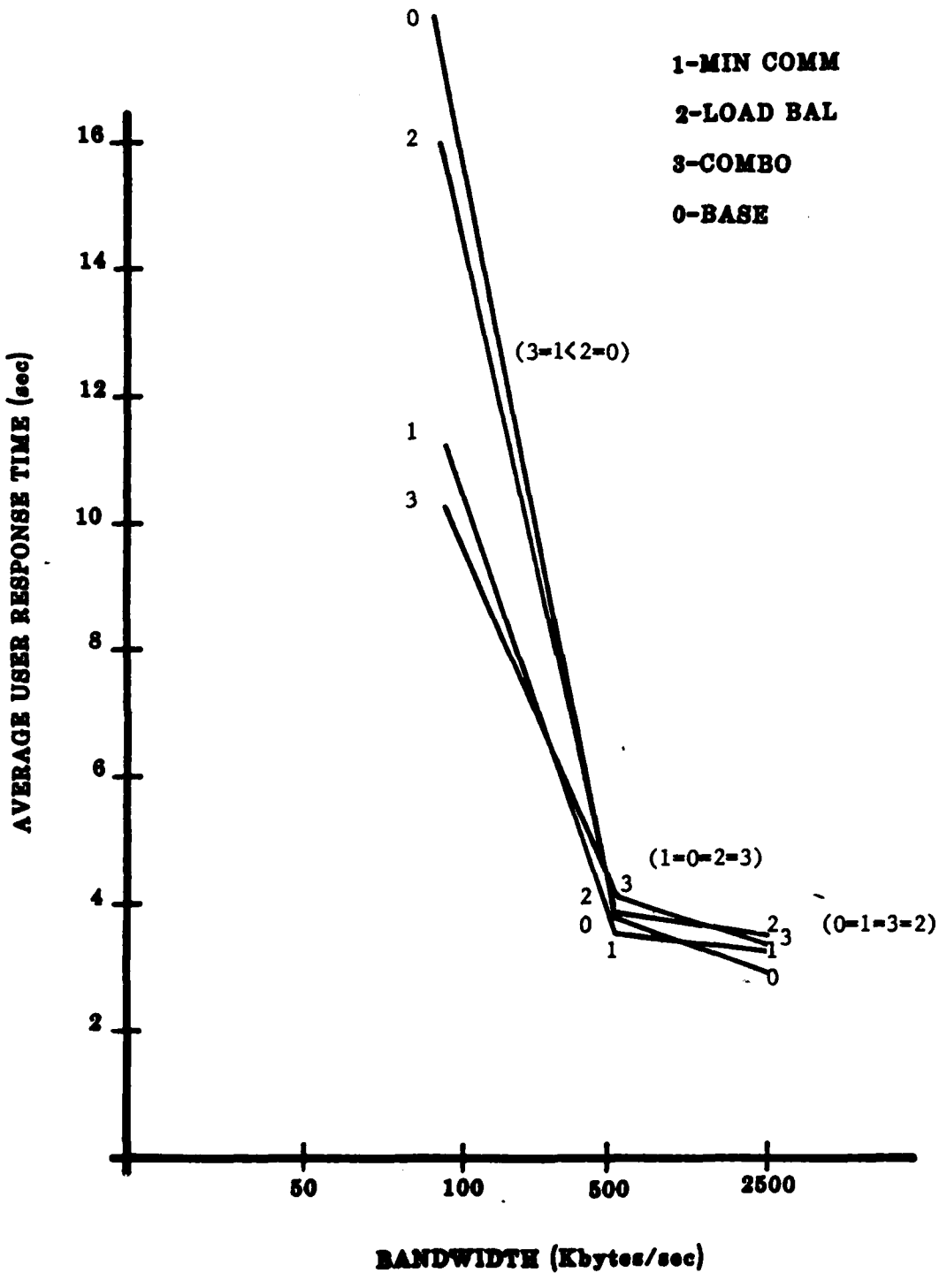


Figure 11. Average User Response Time vs. Bandwidth
For a Uni-Directional Ring Topology with Single File Copies

Table 13. Observed Values and their Means and Standard Deviations of Average User Response Time for a Uni-Directional Ring with Differing Bandwidths

SINGLE FILE COPIES

Communication Bandwidth	Algorithm 1	Algorithm 2	Algorithm 3	Base-Line Algorithm
2,500,000	2.9995	3.1388	3.7383	3.3933
	2.7550	3.4144	3.0571	2.8076
	3.6268	3.6256	3.3503	2.7666
Mean	3.1271	3.3929	3.3819	2.9892
S.D.	0.4497	0.2441	0.3417	0.3506
500,000	3.7525	4.1337	4.8422	3.5052
	3.5868	3.9069	3.7002	3.4310
	3.5596	3.2533	3.5391	4.1201
Mean	3.6330	3.7646	4.0272	3.6854
S.D.	0.1044	0.4571	0.7104	0.3783
100,000	11.0223	14.4428	11.2563	19.2961
	11.2329	15.6451	9.9446	18.9528
	11.5687	17.5719	9.7348	14.4233
Mean	11.2746	15.8886	10.3119	17.5574
S.D.	0.2756	1.5785	0.8246	2.7196

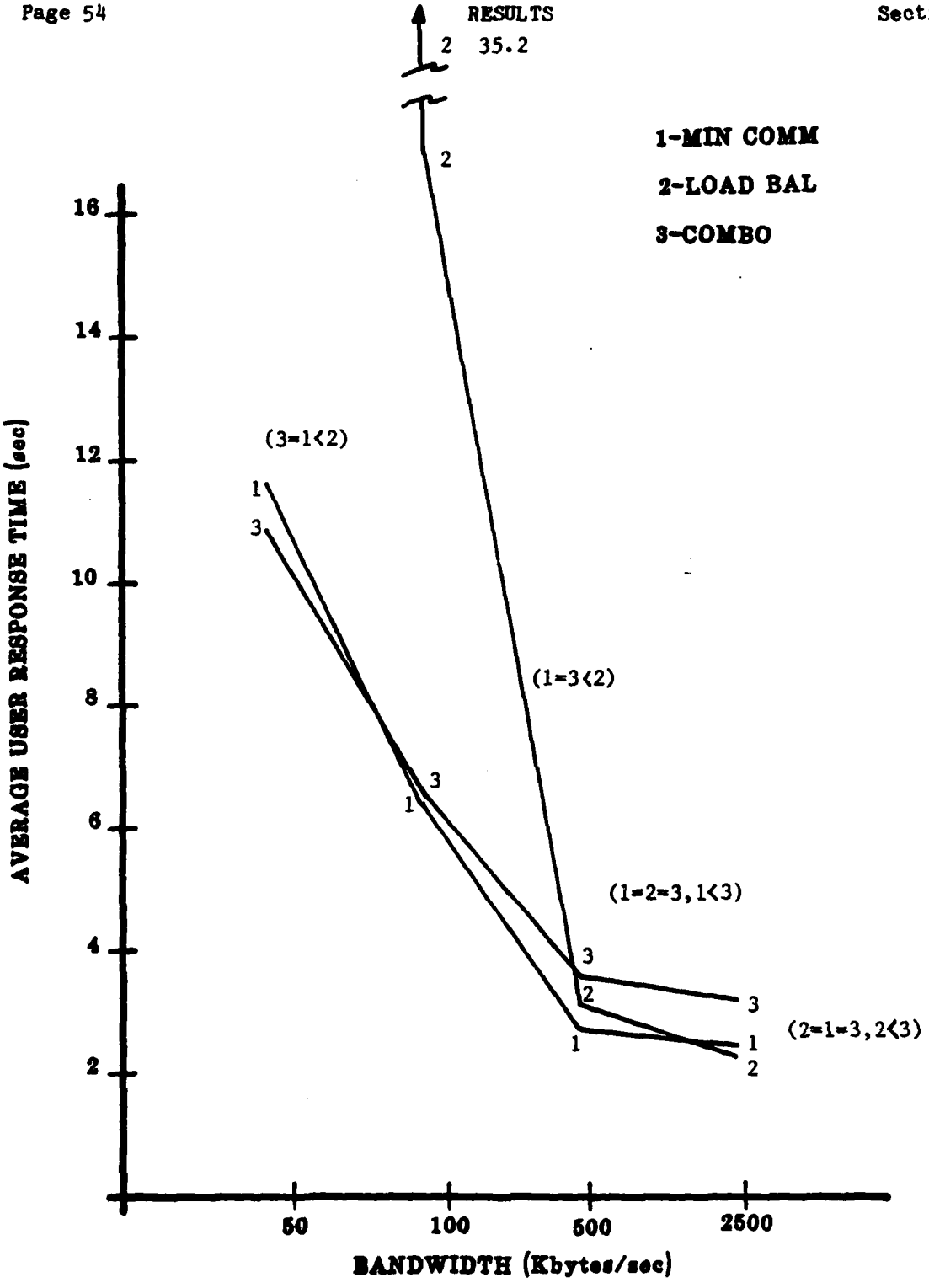


Figure 12. Average User Response Time vs. Bandwidth
For a Uni-Directional Ring Topology with Multiple File Copies

Table 14. Observed Values and their Means and Standard Deviations of Average User Response Time for a Uni-Directional Ring with Differing Bandwidths

MULTIPLE FILE COPIES

Communication Bandwidth	Algorithm 1	Algorithm 2	Algorithm 3
2,500,000	2.3961	2.4059	2.7212
	3.0933	2.4548	3.6224
	1.7686	2.1759	3.4816
Mean	2.4193	2.3455	3.2751
S.D.	0.6627	0.1489	0.4848
500,000	2.7425	3.4942	3.0445
	2.4520	2.2818	3.8912
	2.8140	3.3335	3.6067
Mean	2.6695	3.0365	3.5141
S.D.	0.1917	0.6581	0.4309
100,000	6.0714	15.6113	6.1479
	8.2392	19.1492	6.0659
	5.0157	15.9851	7.2820
Mean	6.4421	16.9152	6.4986
S.D.	1.6434	1.9437	0.6797
50,000	9.0659	35.8079	10.6299
	15.3902	34.8243	12.2840
	10.2521	34.9813	9.7825
Mean	11.5694	35.2045	10.8988
S.D.	3.3616	0.5284	1.2722

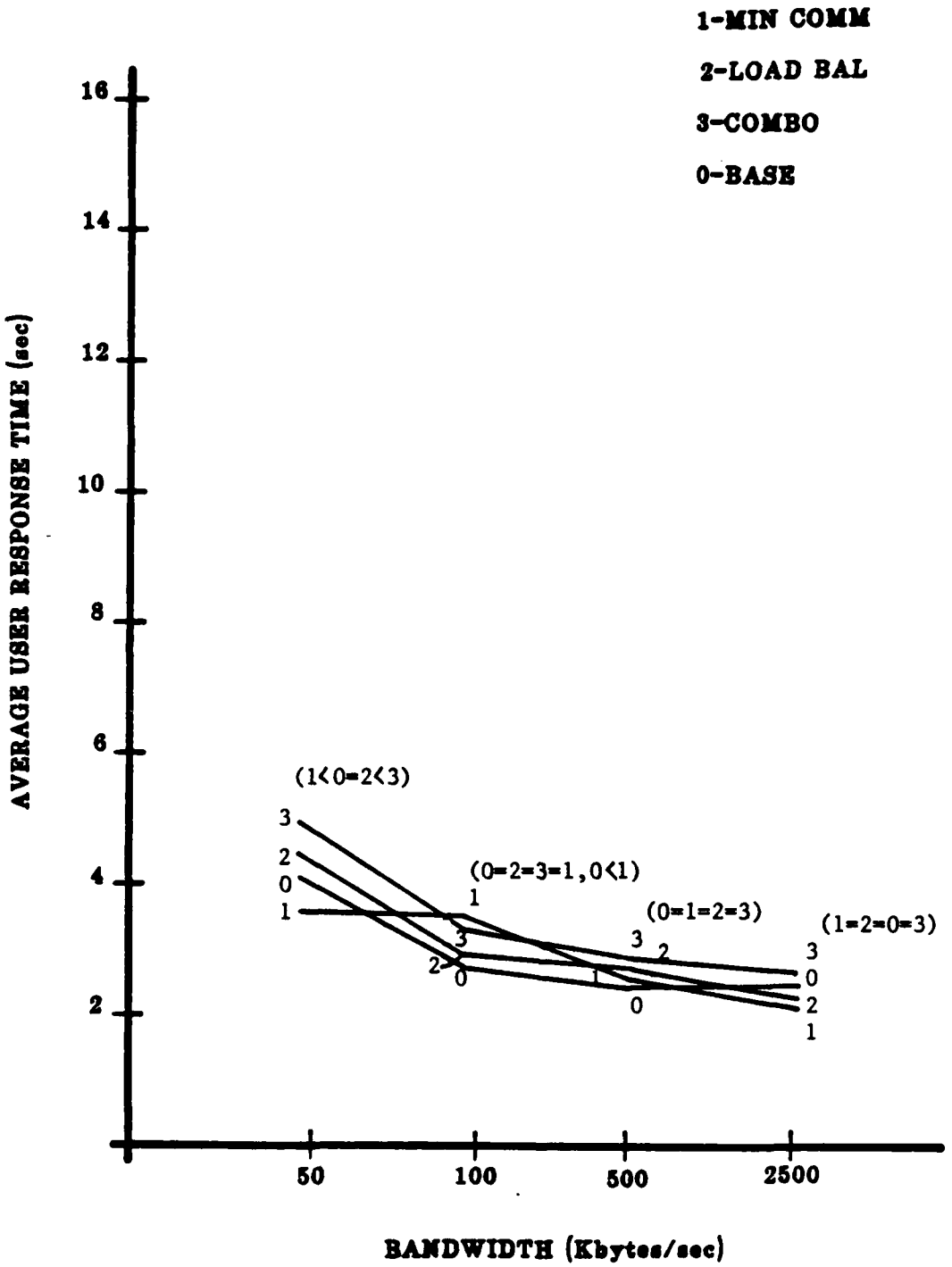


Figure 13. Average User Response Time vs. Bandwidth For a Fully-Connected Network with Single File Copies

Table 15. Observed Values and their Means and Standard Deviations of Average User Response Time for a Fully-Connected Network with Differing Bandwidths

SINGLE FILE COPIES

Communication Bandwidth	Algorithm 1	Algorithm 2	Algorithm 3	Base-Line Algorithm
2.500,000	2.4229	2.4686	2.6168	2.4949
	2.0921	2.4198	2.8591	3.0620
	2.7219	2.4522	2.4422	2.1118
Mean	2.4123	2.4469	2.6394	2.5562
S.D.	0.3150	0.0248	0.2094	0.4781
500,000	2.5087	2.7655	2.6343	2.7300
	2.7867	2.7511	2.5662	2.6845
	2.7119	2.6268	3.0330	2.4050
Mean	2.6691	2.7145	2.7445	2.6065
S.D.	0.1439	0.0763	0.2522	0.1760
100,000	3.5601	2.4517	3.3335	3.0494
	3.6196	2.6988	3.1228	2.9489
	3.2351	3.4204	3.5670	2.4804
Mean	3.4716	2.8570	3.3411	2.8022
S.D.	0.2070	0.5034	0.2222	0.3448
50,000	3.8000	4.6709	5.0410	4.4479
	3.5581	4.6229	5.3598	3.7892
	3.7414	4.3800	4.8789	4.2650
Mean	3.6998	4.5579	5.0966	4.1674
S.D.	0.1262	0.1560	0.2501	0.3400

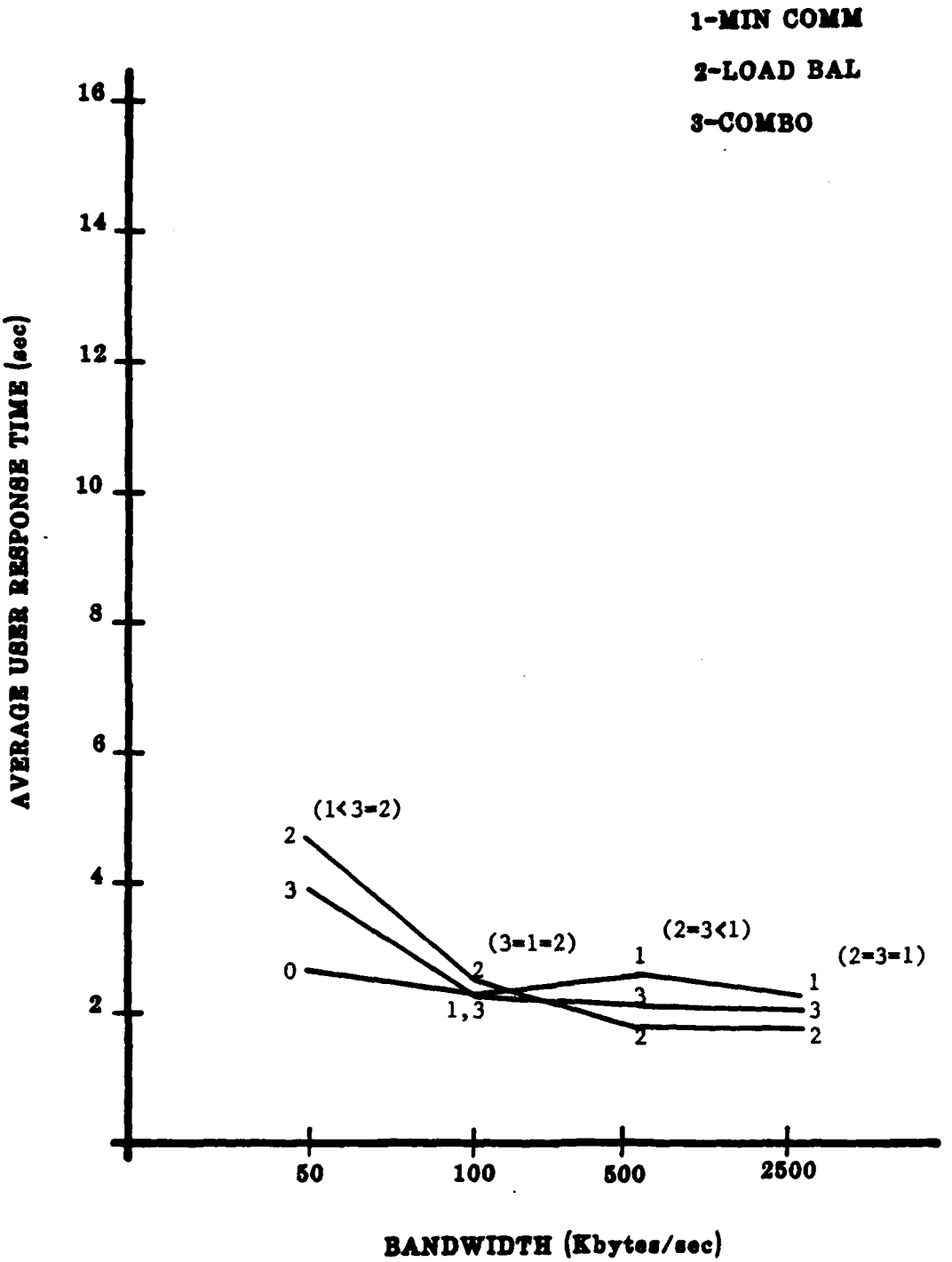


Figure 14. Average User Response Time vs. Bandwidth
For a Fully-Connected Network with Multiple File Copies

Table 16. Observed Values and their Means and Standard Deviations of Average User Response Time for a Fully-Connected Network with Differing Bandwidths

MULTIPLE FILE COPIES

Communication Bandwidth	Algorithm 1	Algorithm 2	Algorithm 3
2,500,000	2.1072	2.0925	1.6654
	1.8646	1.4253	2.4757
	2.4466	1.8772	2.1681
Mean	2.1395	1.7983	2.1031
S.D.	0.2923	0.3405	0.4091
500,000	2.3020	1.8329	2.0019
	2.6882	2.2041	2.2281
	2.6329	1.6586	1.0580
Mean	2.5410	1.8985	2.0960
S.D.	0.2089	0.2786	0.1178
100,000	2.3151	2.4799	2.6109
	2.5545	2.4993	2.0493
	2.3002	2.5446	2.4757
Mean	2.3899	2.5079	2.3786
S.D.	0.1427	0.0332	0.2931
50, 000	2.8611	3.9323	3.4413
	2.5584	4.3347	4.8077
	2.4732	5.6921	3.7123
Mean	2.6309	4.6530	3.9871
S.D.	0.2039	0.9221	0.7235

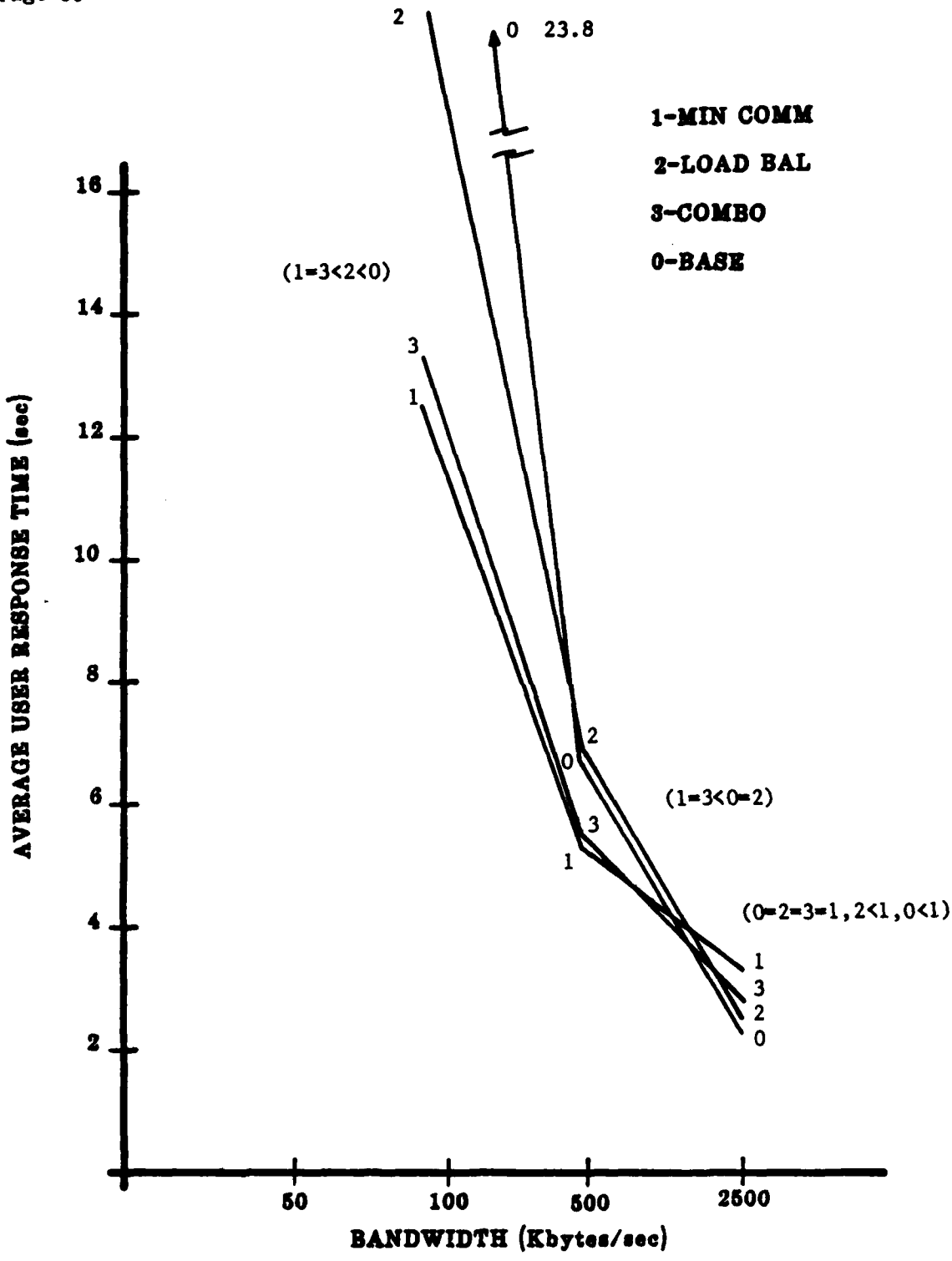


Figure 15. Average User Response Time vs. Bandwidth For a Global Bus Topology with Single File Copies

Table 17. Observed Values and their Means and Standard Deviations of Average User Response Time for a Global Bus Network with Differing Bandwidths

SINGLE FILE COPIES

Communication Bandwidth	Algorithm 1	Algorithm 2	Algorithm 3	Base-Line Algorithm
2,500,000	2.9503	2.6896	2.2995	2.4610
	3.3491	2.6950	3.0113	2.6568
	3.0083	2.1947	2.7258	2.3541
Mean	3.1026	2.5264	2.6789	2.4906
S.D.	0.2155	0.2873	0.3582	0.1535
500,000	4.8270	6.8842	5.5819	6.6263
	5.8156	6.4414	5.2789	6.5438
	5.2871	6.8732	5.4757	6.7659
Mean	5.3099	6.7329	5.4455	6.6453
S.D.	0.4947	0.2525	0.1537	0.1123
100,000	9.1913	18.2875	12.5089	26.3757
	13.0964	16.9397	11.9367	21.9853
	14.7580	17.9922	14.5670	22.9914
Mean	12.3486	17.7398	13.0042	23.7841
S.D.	2.8577	0.7085	1.3833	2.3001

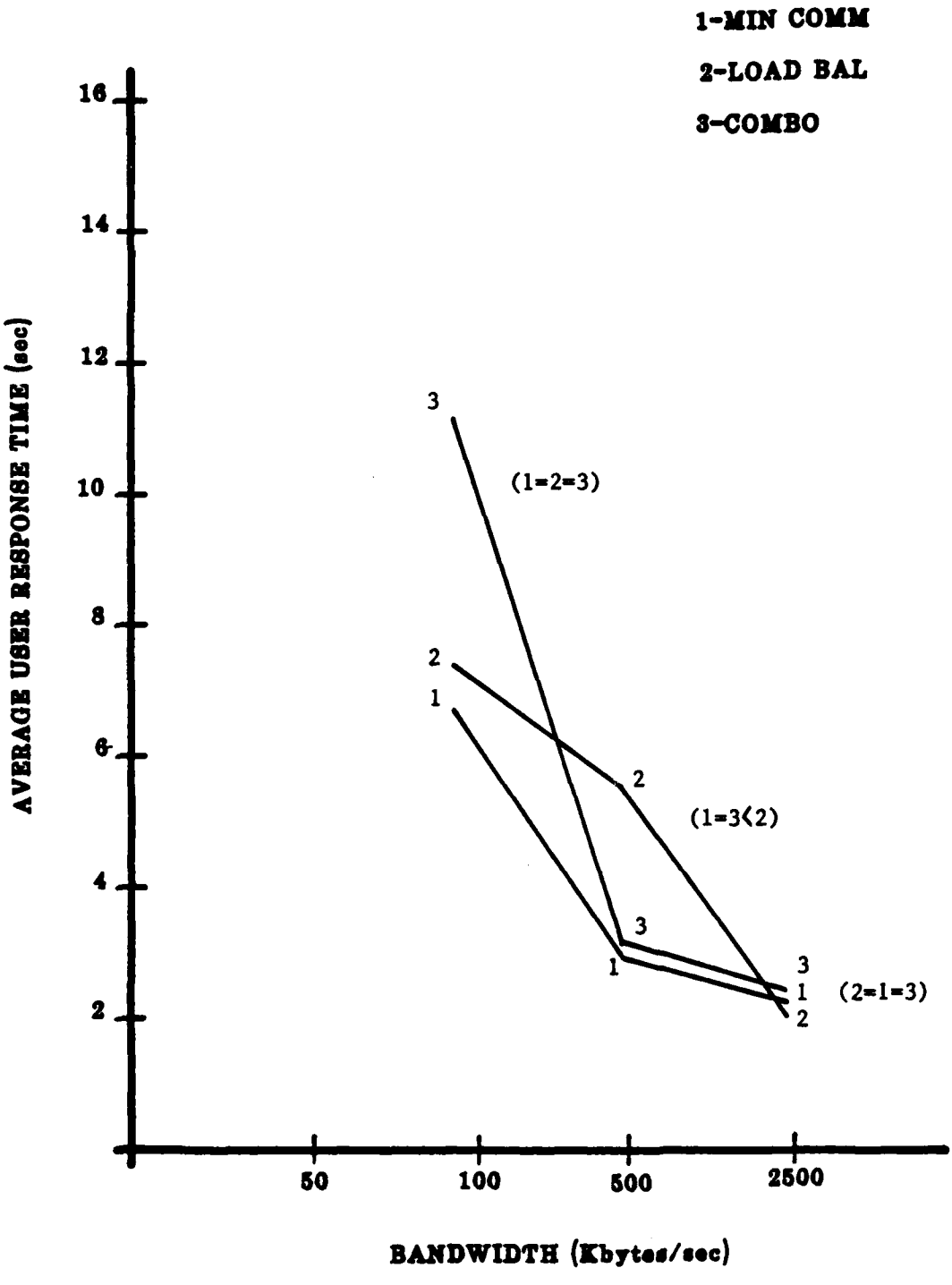


Figure 16. Average User Response Time vs. Bandwidth For a Global Bus Topology with Multiple File Copies

Table 18. Observed Values and their Means and Standard Deviations of Average User Response Time for a Global Bus Network with Differing Bandwidths

MULTIPLE FILE COPIES

Communication Bandwidth	Algorithm 1	Algorithm 2	Algorithm 3
2,500,000	2.1124	2.0967	2.1735
	2.1204	1.8159	2.2969
	2.1260	2.4063	2.3745
Mean	2.1196	2.1063	2.2816
S.D.	0.0068	0.2953	0.1014
500,000	2.8137	5.6248	2.7050
	3.3394	5.0105	3.1913
	2.7159	5.8545	3.0603
Mean	2.9563	5.4966	2.9853
S.D.	0.3353	0.4364	0.2516
100,000	7.5215	8.8744	11.7077
	5.7227	6.6644	12.6682
	6.8325	6.9045	8.8951
Mean	6.6922	7.4811	11.0903
S.D.	0.9076	1.2126	1.9609

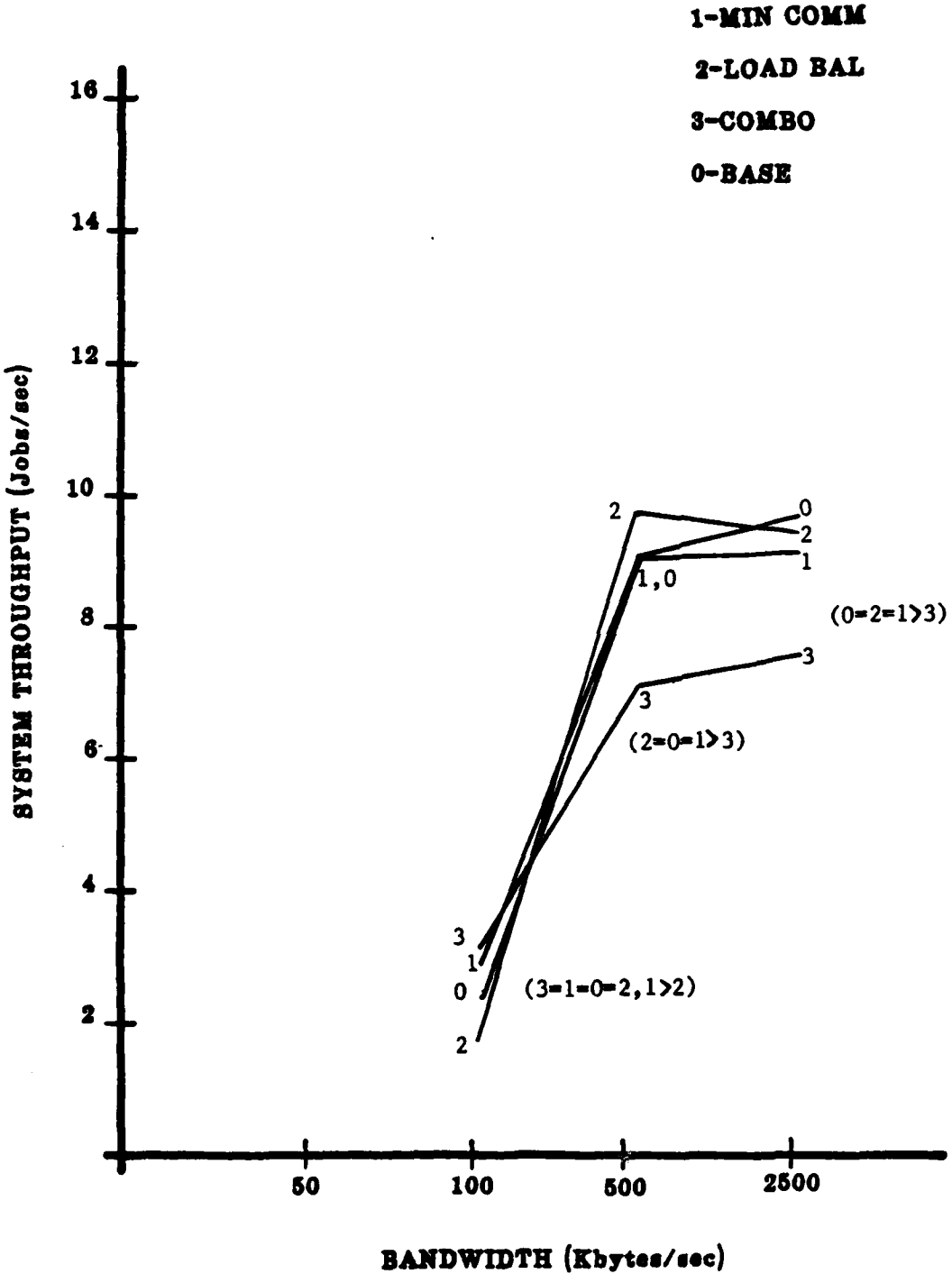


Figure 17. System Throughput vs. Bandwidth
For a Uni-Directional Ring Topology with Single File Copies

Table 19. Observed Values and their Means and Standard Deviations of System Throughput for a Uni-Directional Ring with Differing Bandwidths

SINGLE FILE COPIES

Communication Bandwidth	Algorithm 1	Algorithm 2	Algorithm 3	Base-Line Algorithm
2,500,000	9.5	9.7	7.5	9.7
	9.6	9.3	7.7	9.5
	8.6	9.5	7.7	9.5
Mean	9.2	9.5	7.6	9.6
S.D.	0.6	0.2	0.1	0.1
500,000	8.9	9.2	7.0	9.9
	9.2	8.9	7.2	9.0
	9.6	10.9	7.2	8.6
Mean	9.2	9.7	7.1	9.2
S.D.	0.4	1.1	0.1	0.7
100,000	2.8	1.7	3.1	1.5
	2.9	1.8	2.9	1.8
	2.7	1.8	3.3	3.8
Mean	2.8	1.8	3.1	2.4
S.D.	0.1	0.1	0.2	1.3

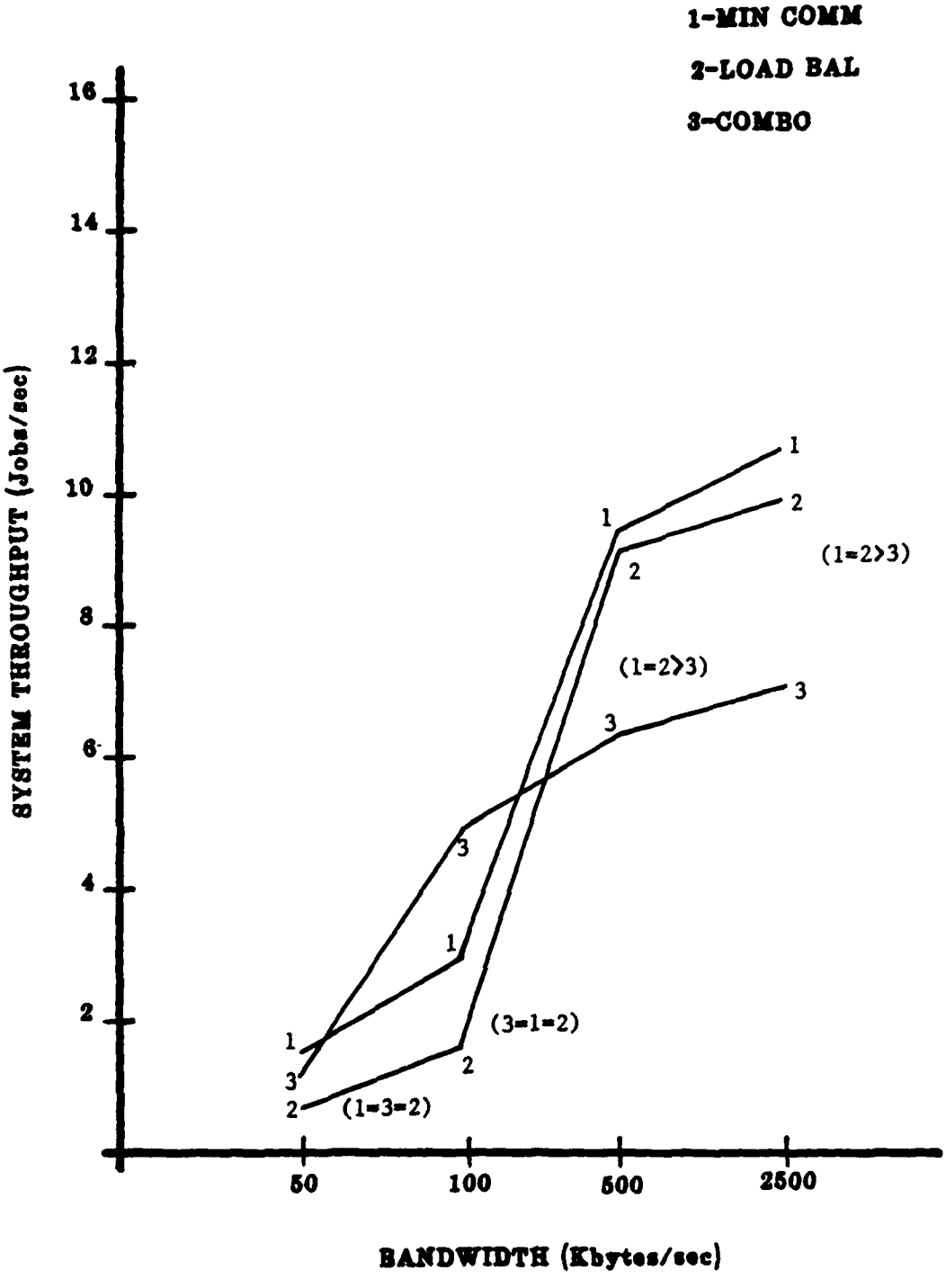


Figure 18. System Throughput vs. Bandwidth
For a Uni-Directional Ring Topology with Multiple File Copies

Table 20. Observed Values and their Means and Standard Deviations of System Throughput for a Uni-Directional Ring with Differing Bandwidths

MULTIPLE FILE COPIES

Communication Bandwidth	Algorithm 1	Algorithm 2	Algorithm 3
2,500,000	12.1	10.8	5.6
	9.8	9.3	8.9
	10.4	10.1	6.8
Mean	10.8	10.1	7.1
S.D.	1.2	0.8	1.7
500,000	10.1	8.9	5.2
	9.9	9.5	6.9
	8.9	9.8	7.3
Mean	9.6	9.4	6.5
S.D.	0.6	0.5	1.1
100,000	2.8	2.6	4.2
	3.9	1.2	4.6
	2.6	1.4	6.1
Mean	3.1	1.7	5.0
S.D.	0.7	0.8	1.0
50,000	1.3	0.6	0.7
	1.7	0.7	1.5
	1.7	1.3	1.6
Mean	1.6	0.9	1.3
S.D.	0.2	0.4	0.5

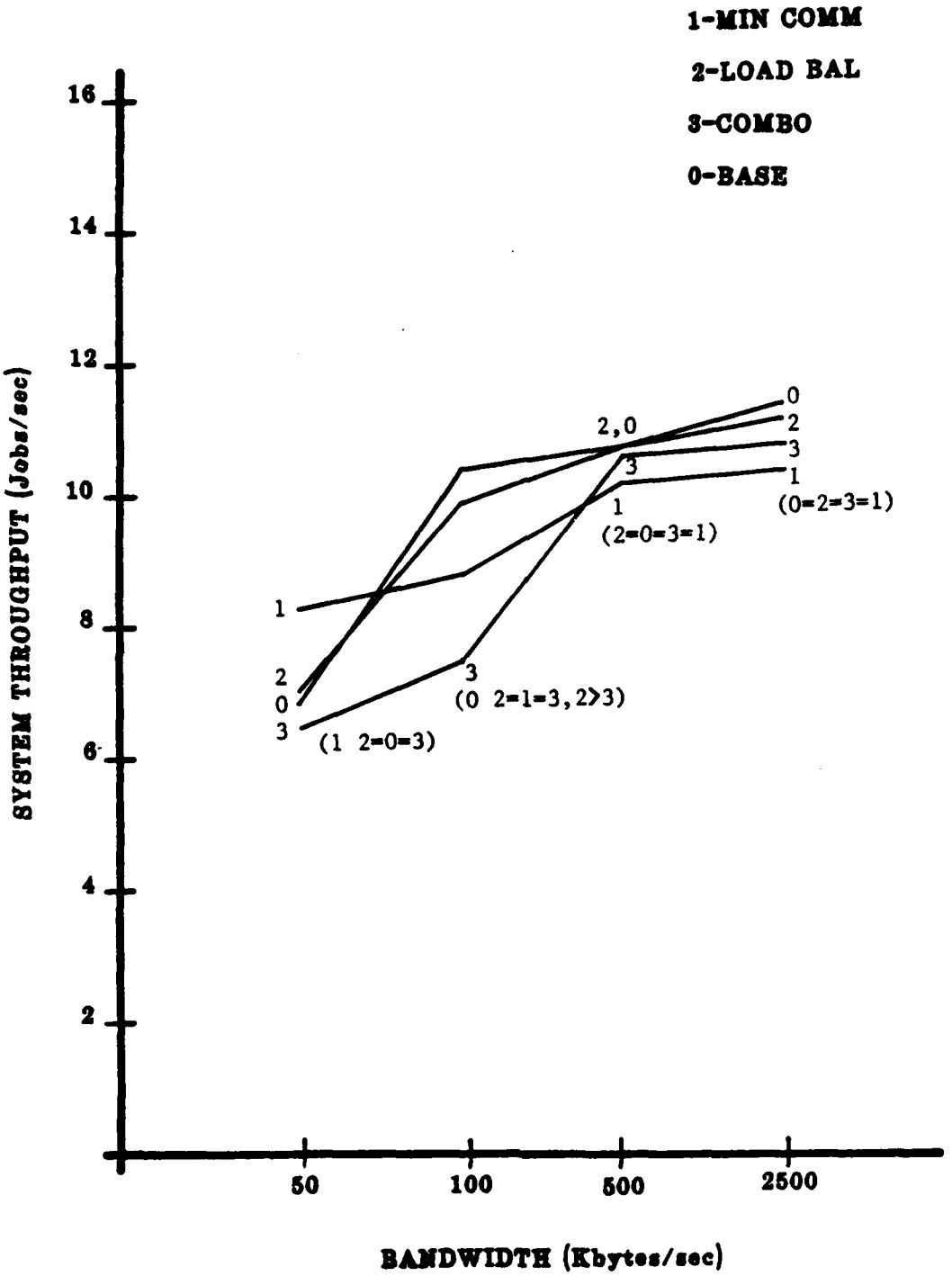


Figure 19. System Throughput vs. Bandwidth
For a Fully-Connected Network with Single File Copies

Table 21. Observed Values and their Means and Standard Deviations of System Throughput for a Fully-Connected Network with Differing Bandwidths

SINGLE FILE COPIES

Communication Bandwidth	Algorithm 1	Algorithm 2	Algorithm 3	Base-Line Algorithm
2,500,000	11.8	11.6	10.5	11.8
	10.0	11.5	10.8	10.6
	9.8	11.1	11.4	12.1
Mean	10.5	11.4	10.9	11.5
S.D.	1.1	0.3	0.5	0.8
500,000	10.5	10.7	10.9	10.0.
	11.1	11.6	10.1	12.7
	9.6	10.3	11.3	9.9
Mean	10.4	10.9	10.8	10.9
S.D.	0.8	0.7	0.6	1.6
100,000	9.9	10.2	8.2	10.5
	9.0	9.9	7.8	10.5
	8.1	10.2	7.1	10.5
Mean	9.0	10.1	7.7	10.5
S.D.	0.9	0.2	0.6	0.0
50,000	8.2	7.7	6.1	7.3
	8.5	6.9	7.3	6.8
	8.7	7.1	6.8	6.9
Mean	8.5	7.2	6.7	7.0
S.D.	0.3	0.4	0.6	0.3

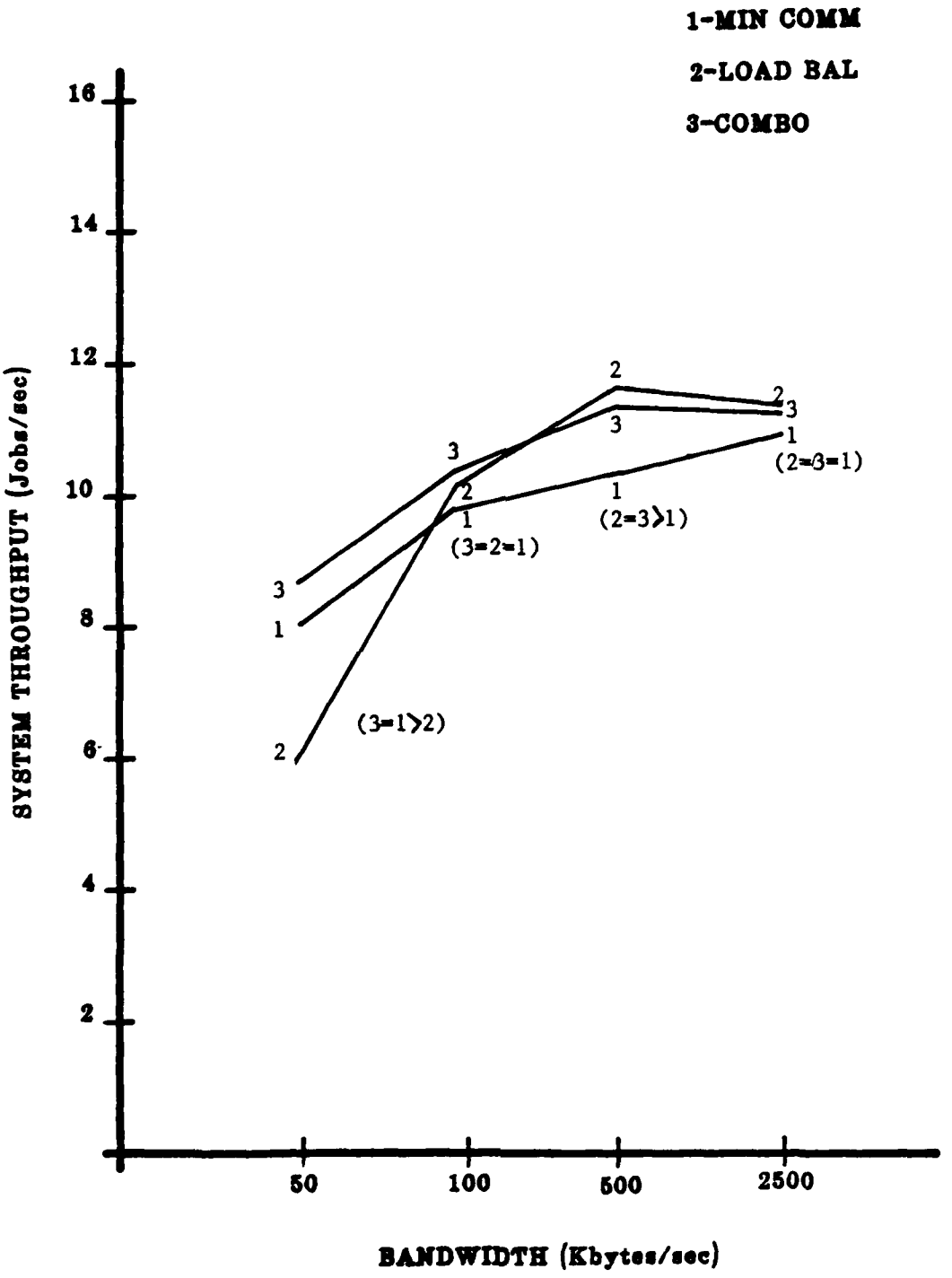


Figure 20. System Throughput vs. Bandwidth
For a Fully-Connected Network with Multiple File Copies

Table 22. Observed Values and their Means and Standard Deviations of System Throughput for a Fully-Connected Network with Differing Bandwidths

MULTIPLE FILE COPIES

Communication Bandwidth	Algorithm 1	Algorithm 2	Algorithm 3
2,500,000	10.6	11.5	11.1
	11.6	10.8	11.6
	10.8	12.1	11.5
Mean	11.0	11.5	11.4
S.D.	0.5	0.7	0.3
500,000	11.2	11.8	11.0
	10.0	11.8	11.6
	10.4	11.9	12.0
Mean	10.5	11.8	11.5
S.D.	0.6	0.1	0.5
100,000	10.4	10.7	11.2
	8.8	10.9	10.4
	10.9	9.3	9.7
Mean	10.0	10.3	10.4
S.D.	1.1	0.9	0.8
50,000	7.8	6.0	9.7
	8.1	6.7	7.5
	8.4	4.8	8.9
Mean	8.1	5.8	8.7
S.D.	0.3	1.0	1.1

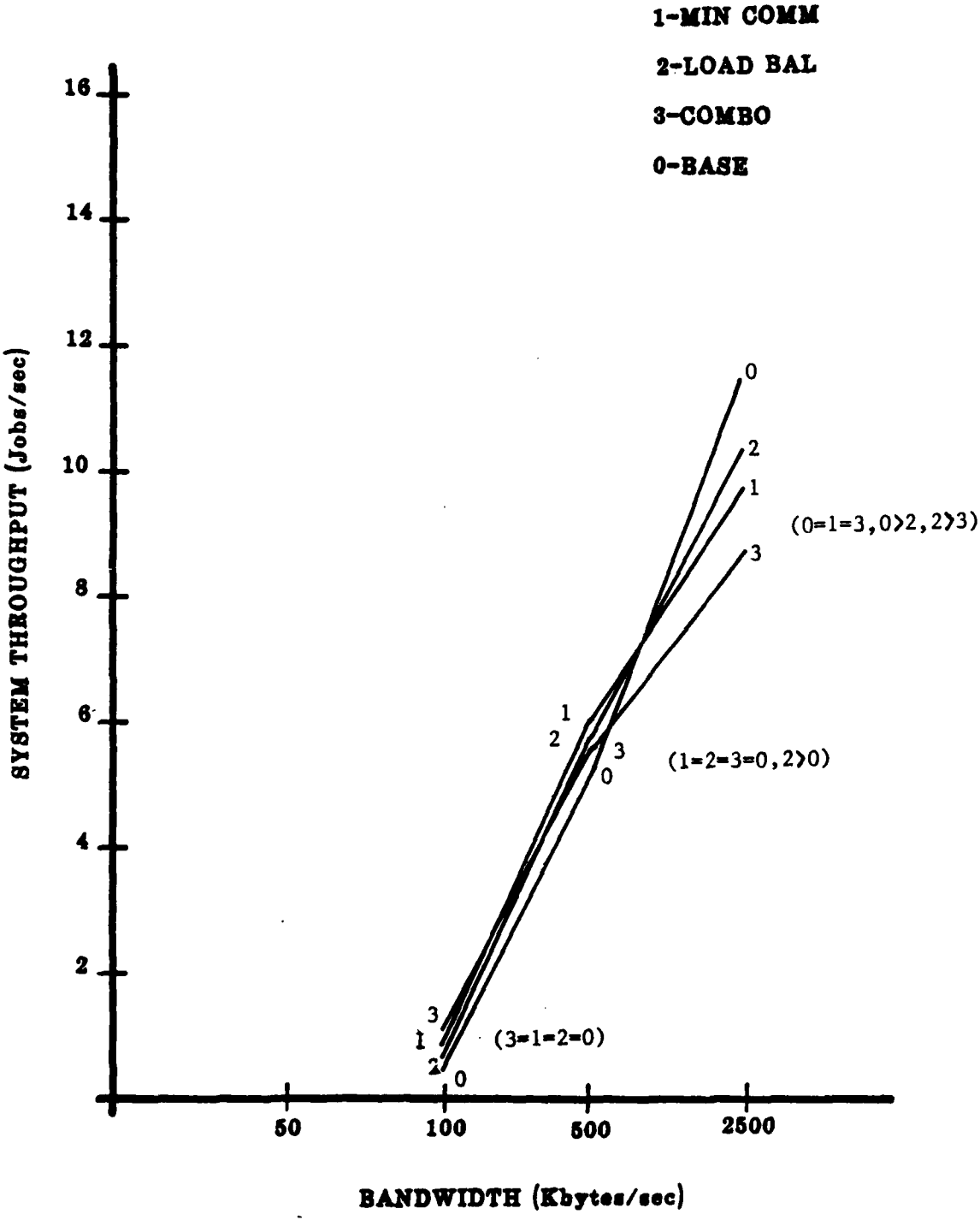


Figure 21. System Throughput vs. Bandwidth
For a Global Bus Topology with Single File Copies

Table 23. Observed Values and their Means and Standard Deviations of System Throughput for a Global Bus Network with Differing Bandwidths

SINGLE FILE COPIES

Communication Bandwidth	Algorithm 1	Algorithm 2	Algorithm 3	Base-Line Algorithm
2,500,000	10.0	10.0	8.8	11.5
	8.8	10.7	8.6	11.0
	10.8	10.6	9.3	12.0
Mean	9.9	10.4	8.9	11.5
S.D.	1.0	0.4	0.4	0.5
500,000	4.9	5.6	5.3	5.3
	6.7	5.8	6.1	5.3
	6.4	5.7	5.5	5.4
Mean	6.0	5.7	5.6	5.3
S.D.	1.0	0.1	0.4	0.1
100,000	0.5	0.5	1.7	0.5
	0.8	0.5	0.6	0.5
	0.7	0.7	0.8	0.6
Mean	0.7	0.6	1.0	0.5
S.D.	0.2	0.1	0.6	0.1

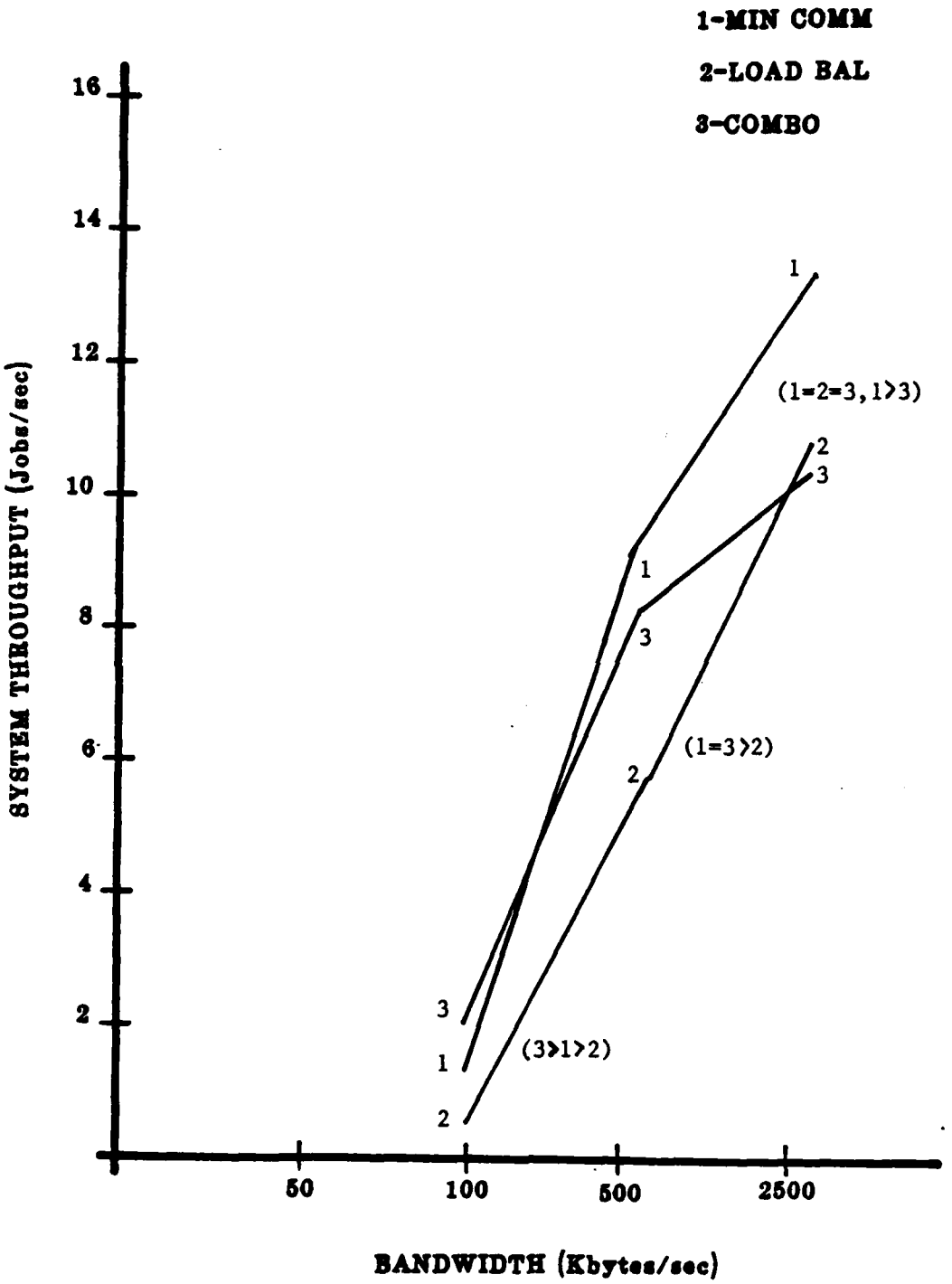


Figure 22. System Throughput vs. Bandwidth
For a Global Bus Topology with Multiple File Copies

Table 24. Observed Values and their Means and Standard Deviations of System Throughput for a Global Bus Network with Differing Bandwidths

MULTIPLE FILE COPIES

Communication Bandwidth	Algorithm 1	Algorithm 2	Algorithm 3
2,500,000	11.1	11.2	10.3
	11.4	11.2	10.4
	11.0	9.6	10.2
Mean	11.2	10.7	10.3
S.D.	0.2	0.9	0.1
500,000	9.2	5.4	7.4
	9.0	5.3	8.5
	9.1	5.7	8.8
Mean	9.1	5.5	8.2
S.D.	0.1	0.2	0.7
100,000	1.4	0.5	1.9
	1.4	0.6	2.0
	1.1	0.5	1.8
Mean	1.3	0.5	1.9
S.D.	0.2	0.1	0.1

Table 25. Values of t for the Different Work Distribution Algorithms with a Uni-Directional Ring

EXPERIMENTS WITH DIFFERENT BANDWIDTHS

Communication Bandwidth	File Copies	T-values t(1,2)	t(1,3)	t(2,3)	t(1,0)	t(2,0)	t(3,0)
		Average User Response Time					
2,500,000	Single	0.0	0.8	0.0	1.4	1.6	1.4
500,000	Single	0.5	1.0	0.5	0.2	0.2	0.7
100,000	Single	5.0*	2.0	5.4*	4.0*	0.9	4.4*
2,500,000	Multiple	0.2	1.8	4.8*			
500,000	Multiple	0.9	3.1*	1.1			
100,000	Multiple	7.1*	0.1	8.8*			
50,000	Multiple	12.0*	0.3	30.6*			
		System Throughput					
2,500,000	Single	1.3	4.6*	14.7*	1.7	0.8	24.5*
500,000	Single	0.8	8.8*	4.1*	0.0	0.7	5.1*
100,000	Single	12.3*	2.3	10.1*	0.5	0.8	0.9
2.500,000	Multiple	0.8	3.1*	2.8*			
500,000	Multiple	0.4	4.3*	4.2*			
100,000	Multiple	2.3	2.7*	4.5*			
50,000	Multiple	2.7*	1.0	1.1			

* Significant at the 95% level (t>2.7).

Note: The t-value t(1,2) compares the performance of Algorithms 1 and 2.

Table 26. Values of t for the Different Work Distribution Algorithms with a Fully-Connected Network

EXPERIMENTS WITH DIFFERENT BANDWIDTHS

Communication Bandwidth	File Copies	T-values t(1.2)	t(1,3)	t(2,3)	t(1,0)	t(2,0)	t(3,0)
		Average User Response Time					
2,500,000	Single	0.2	1.0	1.6	0.4	0.4	0.3
500,000	Single	0.5	0.4	0.2	0.5	1.0	0.8
100,000	Single	2.0	0.7	1.5	2.9*	0.2	2.3
50,000	Single	7.4*	8.6*	3.2*	2.2	1.8	3.8*
2,500,000	Multiple	1.3	0.1	1.0			
500,000	Multiple	3.2*	3.2*	1.1			
100,000	Multiple	1.4	0.1	0.8			
50,000	Multiple	3.7*	3.1*	1.0			
		System Throughput					
2,500,000	Single	1.4	0.6	1.5	1.3	0.2	1.1
500,000	Single	0.8	0.7	0.2	0.5	0.0	0.1
100,000	Single	2.1	2.1	6.6*	2.9*	3.5*	8.1*
50,000	Single	4.5*	4.6*	1.2	6.1*	0.7	0.8
2,500,000	Multiple	1.0	1.2	0.2			
500,000	Multiple	3.7*	2.2	1.0			
100,000	Multiple	0.4	0.5	0.1			
50,000	Multiple	3.8*	0.9	3.4*			

* Significant at the 95% level (t>2.7).

Note: The t-value t(1,2) compares the performance of Algorithms 1 and 2.

Table 27. Values of t for the Different Work Algorithms with a Global Bus Network

EXPERIMENTS WITH DIFFERENT BANDWIDTHS

Communication Bandwidth	File Copies	T-values t(1.2)	t(1.3)	t(2,3)	t(1,0)	t(2,0)	t(3,0)
		Average User Response Time					
2,500,000	Single	2.8*	1.8	0.6	4.0*	0.2	0.8
500,000	Single	4.4*	0.5	7.5*	4.6*	0.5	10.9*
100,000	Single	3.2*	0.4	5.3*	5.4*	4.4*	7.0*
2,500,000	Multiple	0.1	2.8*	1.0			
500,000	Multiple	8.0*	0.1	8.6*			
100,000	Multiple	0.9	3.5*	2.7*			
		System Throughput					
2,500,000	Single	0.8	1.6	4.6*	2.5	3.0*	7.0*
500,000	Single	0.5	0.6	0.4	1.2	4.9*	1.3
100,000	Single	0.8	0.8	1.1	1.5	1.2	1.4
2.500,000	Multiple	0.9	7.0*	0.8			
500,000	Multiple	27.9*	2.2	6.4*			
100,000	Multiple	6.2*	4.6*	17.2*			

* Significant at the 95% level (t>2.7).

Note: The t-value t(1,2) compares the performance of Algorithms 1 and 2.

Table 28. Relative Rankings of the Work Distribution Algorithms

EXPERIMENTS WITH DIFFERENT BANDWIDTHS

Communication Bandwidth	File Copies	Average User Response Time	System Throughput
Uni-Directional Ring			
2,500,000	Single	0=1=3=2	0=2=1>3
500,000	Single	1=0=2=3	2=0=1>3
100,000	Single	3=1<2=0	3=1=0=2 (1>2,3>2)
2,500,000	Multiple	2=1=3 (2<3)	1=2>3
500,000	Multiple	1=2=3 (1<3)	1=2>3
100,000	Multiple	1=3<2	3=1=2 (3>2)
50,000	Multiple	3=1<2	1=3=2
Fully-Connected Network			
2,500,000	Single	1=2=0=3	0=2=3=1
500,000	Single	0=1=2=3	2=0=3=1
100,000	Single	0=2=3=1 (0<1)	0>2=1=3 (2>3)
50,000	Single	1<0=2<3	1>2=0=3
2,500,000	Multiple	2=3=1	2=3=1
500,000	Multiple	2=3<1	2=3>1
100,000	Multiple	3=1=2	3=2=1
50,000	Multiple	1<3=2	3=1>2
Global Bus Network			
2,500,000	Single	0=2=3=1 (2<1,0<1)	0>2=1=3 (2>3,0=1)
500,000	Single	1=3<0=2	1=2=3=0 (2>0)
100,000	Single	1=3<2<0	3=1=2=0
2,500,000	Multiple	2=1=3	1=2=3 (1>3)
500,000	Multiple	1=3<2	1>3>2
100,000	Multiple	1=2=3 (1<3)	3>1>2

Table 29. T-values Comparing Single and Multiple File Copies

EXPERIMENTS WITH DIFFERENT BANDWIDTHS

Communication Bandwidth	T-values t(1,1)	t(2,2)	t(3,3)	t(1,0)	t(2,0)	t(3,0)
Average User Response Time						
Uni-Directional Ring						
2,500,000	1.5	6.3*	0.3	1.3	2.9*	0.8
500,000	7.6*	1.6	1.1	4.2*	1.5	0.5
100.000	5.0*	0.7	6.2*	6.1*	0.3	6.8*
Fully-Connected Network						
2,500,000	1.1	3.3	2.0	1.3	2.2	1.2
500,000	0.9	4.9*	4.0*	2.4	3.7*	2.0
100,000	7.5*	1.2	4.5*	1.9	1.5	1.6
50,000	7.7*	0.2	2.5	6.7*	0.9	0.4
Global Bus Network						
2,500,000	7.9*	1.8	1.8	4.2*	2.0	2.0
500,000	6.8*	4.4*	14.4*	18.1*	4.4*	23.0*
100,000	3.3*	12.7*	1.4	12.0*	10.9*	7.3*
System Throughput						
Uni-Directional Ring						
2,500,000	2.1	1.3	0.5	1.7	1.1	2.5
500,000	1.0	0.4	0.9	0.8	0.4	3.6*
100,000	0.7	0.2	3.2*	0.8	0.8	2.7*
Fully-Connected Network						
2,500,000	0.7	0.2	1.5	0.9	0.0	0.2
500,000	0.2	2.2	1.6	0.4	1.0	0.6
100,000	1.2	0.4	4.7*	0.2	0.4	0.8
50,000	1.6	2.2	2.8*	4.5*	2.0	2.6
Global Bus Network						
2,500,000	2.2	0.5	5.9*	1.0	1.3	4.1*
500,000	5.3*	1.5	5.6*	46.5*	1.5	7.1*
100,000	3.7*	1.2	2.6*	6.2*	0.0	17.1*

* *Significant at the 95% level ($t>2.7$).*

Table 30. Three Components of Average User Response Time for a Uni-Directional Ring with Differing Bandwidths

Communication Bandwidth	Work Request	Algorithm 1	Algorithm 2	Algorithm 3	Base-Line Algorithm
		SINGLE FILE COPIES			
2,500,000	Type 1	2.3182	3.4710	2.5667	2.4963
	Type 2	5.5675	3.5754	4.2263	4.3105
	Type 3	4.4058	4.7747	4.3742	4.4852
500,000	Type 1	2.6641	2.4811	2.3380	3.3489
	Type 2	4.6075	4.2867	4.5527	5.0476
	Type 3	4.5260	4.0529	3.4994	4.2694
100,000	Type 1	9.7359	16.7107	7.4147	13.1220
	Type 2	16.5822	18.6358	14.2912	16.6844
	Type 3	9.5822	18.9394	7.3618	17.3120
		MULTIPLE FILE COPIES			
2,500,000	Type 1	1.7190	1.9511	2.6776	
	Type 2	3.4113	2.8248	4.6304	
	Type 3	4.2863	3.4154	4.0439	
500,000	Type 1	2.3080	2.8160	2.7895	
	Type 2	2.5540	4.0909	4.1838	
	Type 3	7.1407	3.5793	6.3194	
100,000	Type 1	4.4836	15.0635	5.0602	
	Type 2	8.4682	15.7200	7.1522	
	Type 3	10.4836	21.8863	5.5675	
50,000	Type 1	7.5330	27.5689	8.7551	
	Type 2	13.6619	25.8824	9.9043	
	Type 3	24.5168	48.3275	13.0856	

Table 31. Three Components of Average User Response Time for a Fully-Connected Network with Differing Bandwidths

Communication Bandwidth	Work Request	Algorithm 1	Algorithm 2	Algorithm 3	Base-Line Algorithm
		SINGLE FILE COPIES			
2,500,000	Type 1	2.6567	2.0636	2.2973	1.6272
	Type 2	2.6639	2.7491	2.5045	2.3796
	Type 3	3.2115	3.5136	2.9588	4.1994
500,000	Type 1	1.9736	2.0999	2.3889	1.8020
	Type 2	1.7885	2.9877	3.6836	3.1992
	Type 3	4.0519	3.8848	3.7046	2.8465
100,000	Type 1	2.1536	2.5834	2.1635	1.8754
	Type 2	3.5894	4.6555	4.3822	3.1555
	Type 3	2.7794	3.4220	4.2125	2.5681
50,000	Type 1	3.5241	2.8665	3.4160	2.9568
	Type 2	5.3498	5.2231	5.1430	5.7204
	Type 3	4.4175	3.7781	4.8357	4.7286
		MULTIPLE FILE COPIES			
2,500,000	Type 1	1.7981	1.8090	1.9091	
	Type 2	2.3910	2.4707	2.5283	
	Type 3	2.8865	2.5478	2.5240	
500,000	Type 1	1.8619	1.5086	1.4954	
	Type 2	2.6948	2.3332	2.5876	
	Type 3	3.1266	2.0700	3.2725	
100,000	Type 1	1.9717	2.2788	1.6721	
	Type 2	2.2913	3.6009	2.5508	
	Type 3	4.4938	3.6076	3.8572	
50,000	Type 1	4.6669	4.7701	4.2461	
	Type 2	8.5896	7.8170	6.3243	
	Type 3	5.0185	4.8539	4.1716	

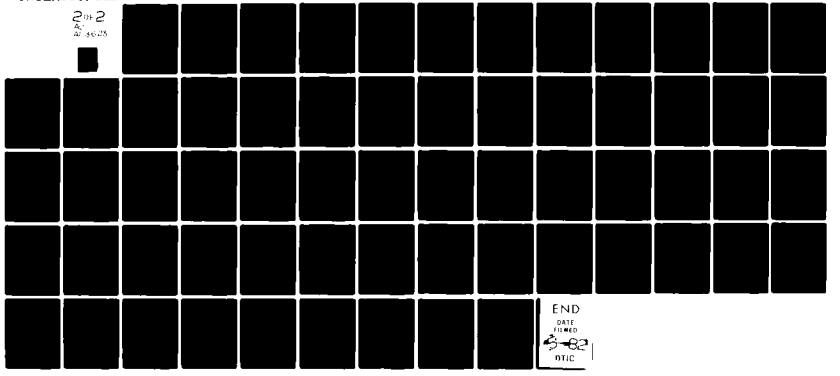
Table 32. Three Components of Average User Response Time for a Global Bus Network with Differing Bandwidths

Communication Bandwidth	Work Request	Algorithm 1	Algorithm 2	Algorithm 3	Base-Line Algorithm
		SINGLE FILE COPIES			
2,500,000	Type 1	2.4353	1.7781	2.0017	1.7023
	Type 2	3.4853	2.6001	3.4129	3.0323
	Type 3	3.7613	2.3385	2.9570	3.4052
500,000	Type 1	3.8182	5.6178	3.5185	5.4039
	Type 2	7.1194	8.7259	8.7826	9.1227
	Type 3	7.8775	6.3136	4.6081	6.3482
100,000	Type 1	14.1544	14.7357	11.6809	19.2606
	Type 2	27.1982	27.6387	28.6445	29.3653
	Type 3	28.1042	28.2542	18.1384	31.2005
		MULTIPLE FILE COPIES			
2,500,000	Type 1	1.8096	2.2295	1.8266	
	Type 2	2.4202	2.1662	3.2293	
	Type 3	2.8784	4.7268	2.6535	
500,000	Type 1	1.9988	4.6688	2.4230	
	Type 2	3.8827	8.2337	3.9967	
	Type 3	4.0550	6.6853	3.8022	
100,000	Type 1	6.1771	8.0282	10.0835	
	Type 2	7.9266	4.1130	15.6483	
	Type 3	16.2503	12.1538	15.4886	

Table 33. Three Components of System Throughput for a Uni-Directional Ring with Differing Bandwidths

Communication Bandwidth	Work Request	Algorithm 1	Algorithm 2	Algorithm 3	Base-Line Algorithm
		SINGLE FILE COPIES			
2,500,000	Type 1	11.5	10.0	10.5	12.5
	Type 2	6.5	9.5	8.0	9.5
	Type 3	7.5	3.5	2.0	5.0
	Power Factor	48.6	39.6	32.4	46.8
500,000	Type 1	13.0	14.5	9.0	10.2
	Type 2	8.5	9.5	7.5	8.0
	Type 3	5.5	2.5	4.0	6.5
	Power Factor	46.8	40.8	36.0	45.6
100,000	Type 1	4.0	2.8	4.3	3.3
	Type 2	2.0	1.0	2.8	1.0
	Type 3	1.8	1.8	2.3	1.3
	Power Factor	13.2	10.2	16.8	9.0
		MULTIPLE FILE COPIES			
2,500,000	Type 1	16.5	7.5	9.5	
	Type 2	9.0	9.0	5.5	
	Type 3	4.0	4.0	5.0	
	Power Factor	46.5	37.5	35.5	
500,000	Type 1	13.5	13.5	11.0	
	Type 2	6.8	8.0	5.0	
	Type 3	4.0	6.5	4.5	
	Power Factor	39.1	49.0	34.5	
100,000	Type 1	5.3	2.5	7.0	
	Type 2	1.8	0.7	2.5	
	Type 3	3.0	0.8	3.0	
	Power Factor	17.9	6.3	21.0	
50,000	Type 1	3.3	1.3	2.5	
	Type 2	0.5	0.1	1.0	
	Type 3	1.0	0.6	1.5	
	Power Factor	7.3	3.3	9.0	

AD-A113 623 GEORGIA INST OF TECH ATLANTA SCHOOL OF INFORMATION A--ETC F/G 9/2
WORK DISTRIBUTION IN A FULLY DISTRIBUTED PROCESSING SYSTEM,(U)
JAN 82 D D SHARP N00014-79-C-0873
UNCLASSIFIED GIT-ICS-82/01 NL



END
DATE FILMED
5-82
DTIC

Table 34. Three Components of System Throughput for a Fully-Connected Network with Differing Bandwidths

Communication Bandwidth	Work Request	Algorithm 1	Algorithm 2	Algorithm 3	Base-Line Algorithm
		SINGLE FILE COPIES			
2,500,000	Type 1	12.0	14.5	13.5	15.0
	Type 2	10.0	11.5	13.0	13.5
	Type 3	5.5	4.0	4.5	3.5
	Power Factor	48.5	49.5	53.0	52.5
500,000	Type 1	13.5	13.5	14.5	13.5
	Type 2	9.5	9.5	11.0	9.0
	Type 3	4.5	6.0	6.5	4.5
	Power Factor	46.0	49.5	56.0	45.0
100,000	Type 1	12.5	13.0	11.5	13.5
	Type 2	10.5	9.0	9.0	12.0
	Type 3	6.5	7.5	5.5	1.5
	Power Factor	53.0	53.5	46.0	42.0
50,000	Type 1	9.5	11.0	10.5	9.5
	Type 2	8.5	7.5	8.5	7.0
	Type 3	4.0	4.5	5.0	5.0
	Power Factor	36.5	39.5	42.5	38.5
		MULTIPLE FILE COPIES			
2,500,000	Type 1	15.0	17.0	16.0	
	Type 2	9.0	9.5	10.0	
	Type 3	5.5	5.5	4.5	
	Power Factor	49.5	53.5	49.5	
500,000	Type 1	14.5	17.5	16.0	
	Type 2	11.0	11.0	9.5	
	Type 3	5.0	2.5	4.0	
	Power Factor	51.5	47.0	47.0	
100,000	Type 1	15.0	15.0	18.0	
	Type 2	10.5	8.0	10.0	
	Type 3	4.5	4.0	2.0	
	Power Factor	49.5	44.5	44.0	
50,000	Type 1	9.0	9.5	9.0	
	Type 2	4.0	5.0	5.5	
	Type 3	5.0	5.0	3.5	
	Power Factor	32.0	34.5	30.5	

Table 35. Three Components of System Throughput for a Global Bus Network with Differing Bandwidths

Communication Bandwidth	Work Request	Algorithm 1	Algorithm 2	Algorithm 3	Base-Line Algorithm
		SINGLE FILE COPIES			
2,500,000	Type 1	13.5	13.0	11.0	16.0
	Type 2	10.0	13.0	10.5	11.5
	Type 3	7.5	2.0	4.0	5.0
	Power Factor	56.0	45.0	44.0	54.0
500,000	Type 1	9.5	7.0	7.5	7.5
	Type 2	6.0	5.5	5.0	4.5
	Type 3	2.0	4.5	3.5	3.5
	Power Factor	27.5	31.5	28.0	27.0
100,000	Type 1	2.3	1.3	2.0	1.2
	Type 2	0.5	0.3	0.5	0.3
	Type 3	0.3	0.2	0.5	0.5
	Power Factor	4.1	2.5	4.5	3.3
		MULTIPLE FILE COPIES			
2,500,000	Type 1	15.5	13.0	14.0	
	Type 2	9.0	9.5	8.0	
	Type 3	6.0	3.5	7.5	
	Power Factor	51.5	42.5	52.5	
500,000	Type 1	14.5	9.0	13.5	
	Type 2	7.0	4.0	6.5	
	Type 3	3.5	3.5	5.5	
	Power Factor	39.0	27.5	42.5	
100,000	Type 1	2.0	0.8	3.3	
	Type 2	1.3	0.3	0.5	
	Type 3	0.5	0.8	1.8	
	Power Factor	6.1	3.8	9.7	

assignment based on the fact that the Type 2 Work Request had more input/output than the Type 1 Work Request, but had less computation than the Type 3 Work Request.

SECTION 5

5.1 Experiments with Different Types of Work Requests

The information in Table 11 summarizes the performance of the different work distribution algorithms in the first set of experiments. An examination of this table shows that Algorithm 1 (minimize communications) had average response times which were significantly lower (better) than or not significantly different from Algorithm 2 (load balancing) or Algorithm 3 (combination) in every test case but one (Test Case 17), where Algorithm 1 was significantly worse than Algorithm 3 and significantly better than Algorithm 2. Algorithm 1 was significantly better than Algorithm 3 in terms of minimizing average user response time in four of the eighteen test cases and was significantly better than Algorithm 2 in ten of the eighteen cases. Except for Test Case 17 described above, Algorithm 1 never had a significantly worse average user response time than Algorithm 3 and never had significantly worse average user response time than Algorithm 2. Algorithm 1 had significantly lower average user response time than the Base-Line Algorithm in three of the nine cases where a comparison could be made and never had significantly worse average user response time than the Base-Line Algorithm.

In terms of system throughput, the pattern of results was much the same. The system throughput for Algorithm 1 was significantly greater (better) than or not significantly different from the throughput of Algorithm 2 and Algorithm 3 in every test case but one (Test Case 3), where Algorithm 2 was significantly better than Algorithm 1, and Algorithm 3 and Algorithm 1 performances were not significantly different from each other. Algorithm 1 was significantly superior to Algorithm 3 in terms of maximizing system throughput in three of the eighteen test cases and was significantly superior to Algorithm 2 in seven of the eighteen cases. Except for Test Case 3, discussed above, Algorithm 1 never had significantly worse system throughput than Algorithm 2 and never had significantly worse throughput than Algorithm 3. Algorithm 1 did not have significantly higher or lower throughput than the Base-Line Algorithm except in Test Case 3 where the Base-Line Algorithm was significantly better.

The information in Table 10 indicates that in four of the nine test cases, Algorithm 1 performed significantly better in terms of average user response time when multiple file copies were available rather than single file copies. Algorithm 1 with multiple file copies also performed significantly better than the Base-Line Algorithm (with single file copies) in those four test cases. This results is not surprising, but it does show that Algorithm 1 can make good use of the information concerning the location of each copy of each file. When average user response time was measured, Algorithm 2 showed similar results in one of the nine test cases and Algorithm 3 showed similar results in three of the nine cases.

In terms of system throughput, in two of the nine cases (in Table 10), Algorithm 1 performed significantly better when multiple file copies were available and Algorithm 1 with multiple file copies performed significantly better than the Base-Line Algorithm (with single file copies). When throughput was measured, Algorithm 2 showed similar results in one of the nine test cases and Algorithm 3 showed similar results in three of the nine cases.

In both the average response time data and the system throughput data of Table 10, the advantage of multiple file copies over single file copies was greater with the global bus topology than with the uni-directional ring topology. There were no significant differences between multiple and single file copies with the fully-connected network. This suggests that a "ceiling" effect took place. With a fully-connected network, the response times were so low with single file copies that there was no room for a significant improvement with multiple file copies.

The advantage of multiple file copies over single file copies also differed by the type of work request. There were more significant differences between single and multiple file copies in Table 10 with Type 2 Work Requests (i1> i2> p >o1 >o2) Than there were with Type 1 Work Requests (i> p >o). There were no significant differences with Type 3 Work Requests (i> p1 ! p2 ! p3 >o).

5.2 Experiments with Different Bandwidths

The information in Table 28 summarizes the performance of the different work distribution algorithms for the second set of experiments. An examina-

tion of Table 28 shows that Algorithm 1 was often significantly better in terms of minimizing average user response time than Algorithm 2 or Algorithm 3 and was rarely significantly worse. Algorithm 1 had significantly lower average user response times than Algorithm 2 in eight of the 22 conditions tested. Algorithm 1 had a significantly higher (worse) average user response time than Algorithm 2 in two of the test cases. Algorithm 1 had a significantly lower (better) average user response time than Algorithm 3 in four of the 22 test cases and had a significantly higher (worse) average user response time than Algorithm 3 in one test case.

Algorithm 1 had a significantly lower (better) average user response time than the Base-Line Algorithm in three of the 11 test cases and had a significantly higher (worse) average user response time in two of the 11 cases. The best performance of Algorithm 1 in terms of minimizing average user response time was at low bandwidths and its worst performance was at very high bandwidths.

The relative superiority of Algorithm 1 (and sometimes also Algorithm 3) can be seen very clearly at the lower bandwidths. For example, Figure 15 (which depicts average user response time with a fully-connected network and single file copies) shows Algorithm 1 to be statistically superior at the lowest bandwidth tested (50,000 bytes/second). With a uni-directional ring topology and single file copies, Figure 11 shows that Algorithms 1 and 3 have lower average user response times than Algorithm 2 or the Base-Line Algorithm at the lowest bandwidth tested (100,000 bytes/second). The same is true with The global bus topology and single file copies (Figure 19). With a uni-directional ring topology and multiple file copies (Figure 13), at the two lowest bandwidths tested (50,000 and 100,000 bytes/second), Algorithms 1 and 3 have significantly lower average user response time than Algorithm 2. With a fully-connected network and multiple file copies (Figure 17), Algorithm 1 has significantly lower response time than either Algorithm 2 or Algorithm 3 at the lowest bandwidth tested (50,000 bytes.second).

In terms of system throughput, the same general pattern of results was observed. Algorithm 1 had significantly higher (better) throughput than Algorithm 2 in six of the 22 test cases and never had significantly lower (worse) throughput than Algorithm 2. Algorithm 1 had a significantly higher

(better) throughput than Algorithm 3 in six of the 22 test cases and had significantly lower (worse) throughput than Algorithm 3 in one test case. Algorithm 1 showed no significant difference in throughput from the Base-Line Algorithm in ten of the eleven test cases and had significantly lower (worse) throughput than the Base-Line Algorithm in one test case. The relative superiority of Algorithm 1 (and sometimes also Algorithm 3) at the lower bandwidths was observed with the throughput data also. An examination of Table 30 shows that in six of the ten test cases Algorithm 1 performed significantly better in terms of average user response time when multiple file copies were available rather than single file copies and Algorithm 1 with multiple file copies performed significantly better than the Base-Line Algorithm (with single file copies). These six cases were all at relatively low bandwidths, showing that Algorithm 1 can effectively utilize multiple file copies to reduce average user response time, at low and moderate bandwidths. At higher bandwidths it appears that a "ceiling" effect took place. Algorith 2 showed similar results in four of the test cases and Algorithm 3 showed similar results for one test case.

In terms of system throughput, in two of the ten cases Algorithm 1 performed significantly better when multiple file copies were available and Algorithm 1 with multiple file copies performed significantly better than the Base-Line Algorithm (with simgle file copies). Algorithm 3 showed similar results in three of the ten test cases.

The advantage of multiple file copies over single file copies differed with the system topology and the bandwidth. This effect was observed in both the average response time data and in the throughput data (Tables 28 and 29). There were more significant differences between the multiple and single file cases with the global bus topology than with the uni-directional ring topology and more with the ring topology than with than with a fully-connected network. This tends to confirm the existence of a "ceiling" effect mentioned above. The significant differences for Algorithm 1 were at the lower bandwidths and the significant differences for Algorithm 2 were at the higher bandwidths. Algorithm 3 had significant differences at both the lower and higher bandwidths.

The information in Tables 30-35 suggests that the relative superiority of Algorithm 1 is due to the way it handles certain types of work requests in the mixture which is presented to it. For example, with a uni-directional ring topology, a bandwidth of 100,000 bytes/second and single file copies (see Table 30), the average user response time was significantly lower for Algorithms 1 and 3 than it was for Algorithms 0 and 2. The difference appears to be due to the better response time for Type 1 and Type 3 Work Requests but not for Type 2 Work Requests. This suggests that the relative ranking of the algorithms can change with a change in the job mix.

The information in Tables 30-35 also provides additional insight into the advantage of multiple file copies over single file copies. A good example is the difference in system throughput between multiple and single file copies for Algorithm 3 with a global bus topology at a bandwidth of 100,000 bytes/second (see Table 35). The throughput is significantly better with multiple file copies and the superiority seems to be due to only the Type 1 and Type 3 Work Requests and not due to the Type 2 Work Requests.

SECTION 6

The objective of this dissertation was to compare the performance of three work distribution algorithms under various test conditions. The following conclusions can be made about the results of the simulation experiment:

1. Algorithm 1 (minimize communications) was better than the other two algorithms in terms of minimizing average user response time and maximizing system throughput over the range of conditions tested. The performance of Algorithm 1 was especially good with low bandwidths, with a global bus topology, and with multiple file copies. Algorithm 1 performed significantly better than the Base-Line Algorithm in many of the test cases, but performed worse than the Base-Line Algorithm in a few test cases.

2. Algorithm 2 (load balancing) did not perform well in comparison to the other algorithms. Algorithm 2 never performed significantly better than Algorithm 1 or the Base-Line Algorithm in any of the test cases and often performed significantly worse. This poor performance is attributed to the well-known problem of updating processor utilization information in a timely manner.

3. Algorithm 3 (combination) sometimes had good performance which matched the good performance of Algorithm 1, but often had bad performance which matched the bad performance of Algorithm 2. If the load-balancing of Algorithm had been more successful, then Algorithm 3 might have performed as well as or better than Algorithm 1.

4. Algorithm 1 (and to a lesser extent, Algorithms 2 and 3) took advantage of multiple file copies to reduce average user response time and to increase system throughput over similar conditions with single file copies. This improvement was especially significant with Type 2 Work Requests (i1> i2> p >o1 >o2), with a bus topology, and with low bandwidths. The design of an efficient work distribution algorithm which can effectively utilize multiple file copies is an important contribution to this area of research. The use of multiple file copies where practicable may turn out

to be the cheapest and easiest way to reduce communications in a distributed processing system and consequently improve system performance.

The lack of significant differences between the average user response times of the three different algorithms in many of the test cases indicates that any one of the algorithms and sometimes no algorithm at all could be used in one of those situations without a significant degradation of performance. This is an important finding because in the real-world conditions change, and a work distribution algorithm which was chosen because it promised good performance in one situation should not degrade performance when conditions change. On the other hand, significant performance differences were obtained for the algorithms based on the specific type of work request used, the network topology, the file redundancy, and the communication bandwidth. Hopefully, the results from this experiment will be used as a framework for additional research on the work distribution problem.

One of the contributions of this experiment was to turn the common notions of "load-balancing" and "minimizing communications" into practical algorithms which can be implemented on a real distributed processing system. A second contribution was to include multiple file copies in the research. A final contribution of the present research was the finding that Algorithm 1 has good performance characteristics over a wide range of test conditions, but especially at low bandwidths.

SECTION 7

The present research contained many interesting results, but several improvements could be made in future research work. These would certainly include additional simulation studies or the implementation of a distributed system. A first recommendation would be to find a better mechanism for updating processor utilization information to facilitate better load balancing. The problem is to receive timely information about existing workloads on the various processors in the network without significantly degrading performance. Performance can be degraded by the increased overhead needed to update the existing workload information due to passing many additional messages between the network nodes.

Several approaches could be taken to solve this problem and might be worthy of future experimentation. Messages containing processor utilization information could be given priority over other messages, processor utilization information could be "piggy-backed" onto other messages, or processor utilization information could be transmitted only when the utilization changes rather than periodically. A solution to this problem would not only improve the performance of the load balancing algorithm, but would probably improve the performance of the combination algorithm as well. Since the combination algorithm was shown to have good performance in some of the test cases in the present experiment, any additional improvement which could be made would be helpful.

Also, many additional experiments could be performed to broaden the scope of the present research by testing additional types of work requests and additional types of network architectures. A mixture of work requests which more accurately modelled a real distributed system workload could be one thing to test. Other network configurations such as a tree structure or a bi-directional ring might be tested. A contention-type message protocol could be tried. Many different types of message transmission mechanisms are possible and could be tested, as well as different, possibly heterogeneous, processors.

SECTION 8

APPENDIX A

PASCAL SOURCE CODE FOR

ALGORITHM NUMBER 1

PASCAL SOURCE CODE FOR ALGORITHM NUMBER 1

```
function rawd (t: tgptr; var node, overhead, stat: integer): integer;
{  Resource allocation and work distribution algorithm number 1.
   Each file may have many copies.  This algorithm tries
   to minimize data communications between nodes.  }

label 9999;

type
   logicalnode = 1..MAXLNODE;
   physicalnode = 1..MAXNODE;
   portnumber = 1..MAXPORT;
   traffictable = array [logicalnode,logicalnode] of real;
   locationtable = array [logicalnode,physicalnode] of boolean;
   speedtable = array [physicalnode,physicalnode] of real;
   timetable = array [physicalnode,physicalnode] of real;
   timearray = array [physicalnode] of real;
   ltimearray = array [logicalnode] of real;
   lnodearray = array [logicalnode] of integer;

var
   i : logicalnode;          {each node in the task graph}
   j : logicalnode;          {each node linked to that node}
   k : logicalnode;          {logical node moved to}
   l : physicalnode;         {from node}
   m : physicalnode;         {to node}
   p : portnumber;           {each output port}
   here, next : physicalnode;
   traffic : traffictable;
   filelocation : locationtable;
   messagespeed : speedtable;
   commtime, movetime : timetable;
   mincommtime, mintotaltime, totmincommtime, sumtraffic : real;
   maxspeed, linktime : real;
   bestnode : physicalnode;
```

```
   newtotcommtime, minmovetime, minnewcommtime, totaltime : timearray;
   processorspeed : timearray;
   fromnode, tonode : lnodearray;
   newcommtime : timearray;
   best : logicalnode;
   thirty, thirtyone : integer;

function cpuexist (n: integer): boolean; extern;
function lqutil (node, src, dest : integer): integer; extern;

begin    { initialization routines }
   thirty := 30000;
   thirtyone := 31000;
   for i := 1 to t^.numnodes do with t^.tgentry[i] do
      begin
         for j := 1 to t^.numnodes do        {initialize traffic}
            traffic[i,j] := 0;
         for p := 1 to numoport do
            begin
               j := oport[p].node;
               if i = j then
                  traffic[i,j] := 0 else
                  if nodetype = PGM
                     then
                        begin
                           traffic[i,j] := thirty;
                           thirty := 15000;
                        end
                     else
                        begin
                           traffic[i,j] := thirtyone;
                           thirtyone := 16000;
                        end;
            end;     {for p}
         thirty := 30000;
```

```
        for l:= 1 to MAXNODE do              {initialize filelocation}
           if (fileavail[l].status = FREE) and (resourceavail[l] <>
                                                  MAXUTIL) then
              filelocation[i,l] := true else
              filelocation[i,l] := false;
     end;        {for i}
  for l := 1 to MAXNODE do                {initialize messagespeed}
     for m := 1 to MAXNODE do
        begin
           if cpuexist(l) and cpuexist(m)
              then
                 if l = m
                    then
                       messagespeed[l,m] := 4000000000 {a large number}
                    else
                       begin
                          next := 0;
                          here := l;
                          linktime := 0;
                          while next <> m do
                             begin
                                next := rtable[here,m];
                                linktime := linktime +
                                          (lqutil(node,here,next) + 1)
                                             / linktabl[1,2].speed;
                                here := next;
                             end;  {while}
                          messagespeed[l,m] := 1 / linktime;
                       end   {else}   {if}
              else
                 messagespeed[l,m] := 1;    {if}
        end;     {for m}  {for l}
  overhead := t^.numnodes * t^.numnodes * 250; {compute rawd overhead}
  { end initialization routines }
```

```
writeln('time = ',curtime.l,'.',curtime.r);

{ begin rawd logic }

for i := 1 to t^.numnodes do with t^.tgentry[i] do
{consider each process in the task graph in order}
 { if procdest.status <> ASSIGNED then  }
{if the process is already assigned then get the next process}
      begin
         for l := 1 to MAXNODE do          {compute processorspeed}
            begin
               if resourceavail[l] = MAXUTIL
                  then
                     processorspeed[l] := 1
                  else
                     processorspeed[l] := 1000000 /
                                          (resourceavail[l] + 1);
            end;     {for l}
         totmincommtime := 0;
         sumtraffic := 0;
         for j := 1 to t^.numnodes do
{for this process, consider all processes linked to it}
{compute the total communication time between this process and all}
   {processes linked to it in the task graph}
            if traffic[i,j] + traffic[j,i] <> 0 then
               begin                       {compute communication time}
                  for l := 1 to MAXNODE do
                     for m := 1 to MAXNODE do
                        commtime[l,m] := maxint;
                  for l := 1 to MAXNODE do
                     if filelocation[i,l] = true then
                        for m := 1 to MAXNODE do
                           if filelocation[j,m] = true then
                              if l = m then
```

```
                                commtime[l,m] := 0 else
                                commtime[l,m] := (traffic[i,j] /
                                                  messagespeed[l,m])
                                                 + (traffic[j,i] /
                                                  messagespeed[m,l]);
                                                    {if}  {if}
                                              {for m}  {if}  {for l}
                    mincommtime := maxint;
                    for l := 1 to MAXNODE do
                       for m := 1 to MAXNODE do
                          if commtime[l,m] < mincommtime
                             then
                                begin
                                   mincommtime := commtime[l,m];
                                   bestnode := l;
                                end      {then}
                             else
                                if (commtime[l,m] = mincommtime) and
                                   (mincommtime <> maxint) and
                                   (processorspeed[l] >
                                      processorspeed[bestnode]) then
                                         bestnode := l;
                                         {if}  {if}  {for m}  {for l}
                       if totmincommtime <> maxint then
                          totmincommtime := totmincommtime + mincommtime;
                       if sumtraffic <> maxint then
                          sumtraffic := sumtraffic + traffic[i,j] +
                                                     traffic[j,i];  {if}
                    end;     {if} {for j}
                 if sumtraffic = 0 then
                    begin
                       maxspeed := 0;
                       for l := 1 to MAXNODE do
                          if (filelocation[i,l] = true) and
                          (processorspeed[l] > maxspeed) then
```

```
                    begin
                       maxspeed := processorspeed[1];
                       bestnode := 1;
                    end;     {if}  {for 1}
              end;        {if}
           for k := 1 to t^.numnodes do
{consider moving this process to the location of each linked process}
{compute total communication time at the new location}
              begin
                 fromnode[k] := 0;
                 tonode[k] := 0;
                 newtotcommtime[k] := 0;
                 totaltime[k] := maxint;
                 minmovetime[k] := maxint;
              end;     {for k}
           for k := 1 to t^.numnodes do
              if traffic[i,k] + traffic[k,i] <> 0 then
                 begin
                    for l := 1 to MAXNODE do
                       for m := 1 to MAXNODE do
                          movetime[l,m] := maxint;
                    for l := 1 to MAXNODE do
                       if filelocation[i,l] = true then
                          for m := 1 to MAXNODE do
                             if filelocation[k,m] = true then
                                if l <> m
                                   then
                                   movetime[l,m] := filesize /
                                                    messagespeed[l,m]
                                   else
                                   movetime[l,m] := 0;  {if}
                                   {if}  {for m}  {if}  {for l}
                    for l := 1 to MAXNODE do
                       for m := 1 to MAXNODE do
                          if movetime[l,m] < minmovetime[k]
```

```
                then
                   begin
                      minmovetime[k] := movetime[l,m];
                      fromnode[k] := l;
                      tonode[k] := m;
                   end   {then}
                else
                   if (movetime[l,m] = minmovetime[k]) and
                      (minmovetime[k] <> maxint) and
                      (processorspeed[l] >
                         processorspeed[fromnode[k]]) then
                      begin
                         fromnode[k] := l;
                         tonode[k] := m;
                      end;  {if}  {if}  {for m}  {for l}
for j := 1 to t^.numnodes do
   if (j <> k) and (traffic[k,j]
                  +  traffic[j,k] <> 0) then
      begin
         for m:= 1 to MAXNODE do
            newcommtime[m] := 0;
         for m := 1 to MAXNODE do
            if filelocation[j,m] = true then
               if (tonode[k] <> m) and
                  (tonode[k] <> 0)
                  then
                     begin
                  newcommtime[m] := (traffic[i,j] /
                           messagespeed[tonode[k],m])
                           + (traffic[j,i] /
                           messagespeed[m,tonode[k]]);
                     end
                  else
                  newcommtime[m] := 0; {if}  {if}
                                       {for m}
```

```
                        minnewcommtime[j] := maxint;
                        for m := 1 to MAXNODE do
                           if newcommtime[m] < minnewcommtime[j]
                              then
                                 minnewcommtime[j] :=
                                          newcommtime[m];
                        newtotcommtime[k] := newtotcommtime[k] +
                                             minnewcommtime[j];
                     end;        {if}  {for j}
                totaltime[k] := minmovetime[k] + newtotcommtime[k];
             end;        {if}  {for k}
          mintotaltime := maxint;
          for k := 1 to t^.numnodes do
   {compare these communication times and chose the minimum}
             if totaltime[k] < mintotaltime then
                begin
                   mintotaltime := totaltime[k];
                   best := k;
                end;        {if}  {for k}
          if (mintotaltime = maxint) and (totmincommtime = maxint)
             then
                begin
   {if no assignment can be made, return error indicator}
                   stat := ER;
                   rawd := ER;
                   writeln(procname,'ER');
                   goto 9999;
                end      {then}
             else
                begin
   {if file is not to be moved, indicate selected location for execution}
                   if (totmincommtime <= mintotaltime) or
                      (nodetype <> PGM)
                      then
                         begin
```

```
                        procdest.node := bestnode;
                        procutil.node := bestnode;
                        writeln(procname,bestnode);
                     end      {then}
                  else
                     begin
   {if file is to be moved, indicate from and to locations}
                        procdest.node := tonode[best];
                        procutil.node := fromnode[best];
                        writeln(procname,fromnode[best],
                                         tonode[best]);
                        for l := 1 to MAXNODE do
                                    {update filelocation}
                           if l = tonode[best] then
                              filelocation[i,l] := true else
                              filelocation[i,l] := false;
                                    {if}  {for l}
                     end;     {else}  {if}
               { procdest.status := ASSIGNED; }
            end;        {else}  {if}
      end;        {if}  {for i}
   stat := OK;
   rawd := OK;
9999: end;        {rawd}
```

APPENDIX B

PASCAL SOURCE CODE FOR

ALGORITHM NUMBER 2

PASCAL SOURCE CODE FOR ALGORITHM NUMBER 2

```
function rawd (t: tgptr; var node, overhead, stat: integer): integer;
{  Resource allocation and work distribution algorithm number 2.
   Each file may have many copies.  This algorithm tries to
   maximize processor utilization by assigning new processes to
   processors that are not heavily utilized.}

label 9999;

type
   logicalnode = 1..MAXLNODE;
   physicalnode = 1..MAXNODE;
   timearray = array [physicalnode] of real;
   timetable = array [physicalnode,physicalnode] of real;
   speedarray = array [physicalnode] of real;
   speedtable = array [physicalnode,physicalnode] of real;
   locationtable = array [physicalnode,physicalnode] of boolean;

var
   i : logicalnode;          {each node in the task graph}
   l : physicalnode;         {from node}
   m : physicalnode;         {to node}
   here, next : physicalnode;
   fromnode, tonode : physicalnode;
   mintotal, linktime : real;
   numbinstructions : integer;
   processtime, movetime : timearray;
   totaltime : timetable;
   processorspeed : speedarray;
   messagespeed : speedtable;
   filelocation : locationtable;

function cpuexist (n: integer): boolean; extern;
function lqutil (node, src, dest : integer): integer; extern;
```

```
begin    { initialization routines }
   for i := 1 to t^.numnodes do with t^.tgentry[i] do
      for l := 1 to MAXNODE do          {initialize filelocation}
         if (fileavail[l].status = FREE) and (resourceavail[l] <>
                                              MAXUTIL) then
            filelocation[i,l] := true else
            filelocation[i,l] := false;   {if}  {for l}  {for i}
   for l := 1 to MAXNODE do               {initialize messagespeed}
      for m := 1 to MAXNODE do
         begin
            if cpuexist(l) and cpuexist(m)
               then
                  if l = m
                     then
                        messagespeed[l,m] := 4000000000
                     else
                        begin
                           next := 0;
                           here := l;
                           linktime := 0;
                           while next <> m do
                              begin
                                 next := rtable[here,m];
                                 linktime := linktime +
                                             (lqutil(node,here,next) + 1)
                                                / linktabl[1,2].speed;
                                 here := next;
                              end;  {while}
                           messagespeed[l,m] := 1 / linktime;
                        end   {else}   {if}
               else
                  messagespeed[l,m] := 1;    {if}
         end;     {for m}  {for l}
   overhead := t^.numnodes * 250;
   { end initialization routines }
```

```
writeln('time = ',curtime.l,'.',curtime.r);

{ begin rawd logic }

for i := 1 to t^.numnodes do with t^.tgentry[i] do
{consider each process in the task graph in order}
   begin
      for l := 1 to MAXNODE do
         begin
            if resourceavail[l] = MAXUTIL
               then
                  processorspeed[l] := 1
               else
                  processorspeed[l] := 1000000 /
                                       (resourceavail[l] + 1);
         end;     {for l}
      if nodetype = PGM then
         numbinstructions := 90000 else
         numbinstructions := 1;                    {default}
   {  if procdest.status <> ASSIGNED then  }
{if the process is already assigned then get the next process}
         for l := 1 to MAXNODE do      {for each process}
            if (filelocation[i,l] = true)
               then
{compute the processing time for this process at all present locations}
                  for m := 1 to MAXNODE do
                     if (resourceavail[m] <> MAXUTIL) then
                     begin
{consider moving the process to another location}
{calculte the sum of move time plus processing time}
                        processtime[m] := numbinstructions /
                                          processorspeed[m];
                        if m = l
                           then
```

```
                        movetime[m] := 0
                     else
                        movetime[m] := filesize /
                                       messagespeed[1,m];
                  if resourceavail[m] <> MAXUTIL
                     then
                        totaltime[1,m] := processtime[m] +
                                          movetime[m]
                     else
                        totaltime[1,m] := maxint;  {if}
               end      {if}  {for m}  {then}
               else
                  totaltime[1,m] := maxint   {if}  {for m}
                                             {then}
            else
               for m := 1 to MAXNODE do
                  totaltime[1,m] := maxint; {for m}  {if}  {for 1}
      mintotal := maxint;
      for 1 := 1 to MAXNODE do
         for m := 1 to MAXNODE do
{chose the minimum total time}
            if totaltime[1,m] < mintotal then
               begin
                  mintotal := totaltime[1,m];
                  fromnode := 1;
                  tonode := m;
               end;     {if}  {for m}  {for 1}
      if mintotal = maxint
         then
            begin
{if no assignment can be made, return error indicator}
               stat := ER;
               rawd := ER;
               writeln(procname,'ER');
               goto 9999;
```

```
            end      {then}
         else
            begin                              {assign process}
   {indicate selected location as to location}
   {if file is to be moved, then from location is different}
               procdest.node := tonode;
               procutil.node := fromnode;
               writeln(procname,fromnode,tonode);
            {  procdest.status := ASSIGNED; }
               resourceavail[tonode] := resourceavail[tonode] + 1;
                                    {update processor availability}
               for l := 1 to MAXNODE do      {update filelocation}
                  if l = tonode then
                     filelocation[i,l] := true else
                     filelocation[i,l] := false;   {if}  {for l}
            end;     {else}  {if}
      end;        {for i}
   stat := OK;
   rawd := OK;
9999: end;        {rawd}
```

APPENDIX C

PASCAL SOURCE CODE FOR

ALGORITHM NUMBER 3

PASCAL SOURCE CODE FOR ALGORITHM NUMBER 3

```
%INCLUDE '<ub>sharp>simulator>sim_def.i';
{$E+}
var
   %INCLUDE '<ub>tim>radc>simulator>stdmods>sim_globalvars.i';
function rawd (t: tgptr; var node, overhead, stat: integer): integer;
{  Resource allocation and work distribution algorithm number 3.
   Each file may have many copies.  This algorithm tries
   to minimize the total of copy time, communication time,
   and process time.  }

label 9999;

type
   logicalnode = 1..MAXLNODE;
   physicalnode = 1..MAXNODE;
   portnumber = 1..MAXPORT;
   traffictable = array [logicalnode,logicalnode] of real;
   locationtable = array [physicalnode,physicalnode] of boolean;
   speedtable = array [physicalnode,physicalnode] of real;
   speedarray = array [physicalnode] of real;
   timearray = array [physicalnode] of real;
   nodearray = array [physicalnode] of integer;

var
   i : logicalnode;              {each node in the task graph}
   j : logicalnode;              {each node linked to that node}
   l : physicalnode;             {from node}
   m : physicalnode;             {to node}
   p : portnumber;               {each output port}
   here, next : physicalnode;
   traffic : traffictable;
   filelocation : locationtable;
   processorspeed : speedarray;
   messagespeed : speedtable;
```

```
   movetime, commtime, totaltime : timearray;
   minmovetime, processstime, totmincommtime, mintottime : real;
   mincommtime, linktime : real;
   best : physicalnode;
   numbinstructions, thirty, thirtyone : integer;
   fromnode, tonode : nodearray;

function cpuexist (n: integer): boolean; extern;
function lqutil (node, src, dest : integer): integer; extern;

begin    { initialization routines }
   thirty := 30000;
   thirtyone := 31000;
   for i := 1 to t^.numnodes do with t^.tgentry[i] do
      begin
         for j := 1 to t^.numnodes do          {initialize traffic}
            traffic[i,j] := 0;
         for p := 1 to numoport do
            begin
               j := oport[p].node;
               if i = j then
                  traffic[i,j] := 0 else
                     if nodetype = PGM
                     then
                        begin
                           traffic[i,j] := thirty;
                           thirty := 15000;
                        end
                     else
                        begin
                           traffic[i,j] := thirtyone;
                           thirtyone := 16000;
                        end;
            end;     {for p}
         thirty := 30000;
```

```
        for l := 1 to MAXNODE do            {initialize filelocation}
           if (fileavail[l].status = FREE) and (resourceavail[l] <>
                                                MAXUTIL) then
              filelocation[i,l] := true else
              filelocation[i,l] := false;
     end;        {for i}
  for l := 1 to MAXNODE do                  {initialize messagespeed}
     for m := 1 to MAXNODE do
        begin
           if cpuexist(l) and cpuexist(m)
              then
                 if l = m
                    then
                       messagespeed[l,m] := 4000000000  {a large number}
                    else
                       begin
                          next := 0;
                          here := l;
                          linktime := 0;
                          while next <> m do
                             begin
                                next := rtable[here,m];
                                linktime := linktime +
                                            (lqutil(node,here,next) + 1)
                                               / linktabl[1,2].speed;
                                here := next;
                             end;  {while}
                          messagespeed[l,m] := 1 / linktime;
                       end   {else}   {if}
              else
                 messagespeed[l,m] := 1;    {if}
        end;     {for m}  {for l}
  overhead := t^.numnodes * t^.numnodes * 250;
  { end initialization routines }
```

```
writeln('time = ',curtime.l,'.',curtime.r);

{ begin rawd logic }

for i := 1 to t^.numnodes do with t^.tgentry[i] do
{consider each process in the task graph in order}
{  if procdest.status <> ASSIGNED then  }
{if the process is already assigned then get the next process}
      begin
         for l := 1 to MAXNODE do            {compute processorspeed}
            begin
               if resourceavail[l] = MAXUTIL
                  then
                     processorspeed[l] := 1
                  else
                     processorspeed[l] := 1000000 /
                                          (resourceavail[l] + 1);
            end;     {for l}
         if nodetype = PGM then
            numbinstructions := 90000 else
            numbinstructions := 1;          {default}
         for m := 1 to MAXNODE do
            totaltime[m] := 0;
         for m := 1 to MAXNODE do
            if (resourceavail[m] <> MAXUTIL) then
            begin
               if filelocation[i,m] = true
                  then
                     begin
                        minmovetime := 0;
                        fromnode[m] := m;
                        tonode[m] := m;
                     end      {then}
                  else
```

```
                    begin
   {consider moving this process to each other node}
   {compute the total of communication time, move time and processing time}
                       for l := 1 to MAXNODE do
                          movetime[l] := maxint;
                       for l := 1 to MAXNODE do
                          if (filelocation[i,l] = true) and (l <> m)
                             then
                                movetime[l] := filesize /
                                               messagespeed[l,m];
                                               {if}  {for l}
                          minmovetime := maxint;
                          for l := 1 to MAXNODE do
                             if (movetime[l] < minmovetime) then
                                begin
                                   minmovetime := movetime[l];
                                   fromnode[m] := l;
                                   tonode[m] := m;
                                end;     {if}  {for l}
                       end;        {else}  {if}
                    processstime := numbinstructions / processorspeed[m];
                    totmincommtime := 0;
                    for j := 1 to t^.numnodes do
                       if (traffic[i,j] + traffic[j,i]) <> 0 then
                          begin             {chose min total comm time}
                             for l := 1 to MAXNODE do
                                commtime[l] := maxint;
                             for l := 1 to MAXNODE do
                                if (filelocation[j,l] = true) then
                                   if l <> m
                                      then
                                      commtime[l] := (traffic[i,j] /
                                                     messagespeed[m,l]) +
                                                     (traffic[j,i] /
                                                     messagespeed[l,m])
```

```
                                    else
                                    commtime[l] := 0;    {if}
                                                 {if}  {for l}
                           mincommtime := maxint;
                           for l := 1 to MAXNODE do
                              if commtime[l] < mincommtime then
                                 mincommtime := commtime[l];
                           totmincommtime := totmincommtime  +
                                             mincommtime;
                        end;     {if}  {for j}
                  if minmovetime <> maxint
                     then
                        totaltime[m] := minmovetime + processtime +
                                        totmincommtime
                     else
                        totaltime[m] := maxint;    {if}
               end         {then}
               else
                  totaltime[m] := maxint;    {if}  {for m}
            mintottime := maxint;
            for m := 1 to MAXNODE do
   {select the minimum total time}
               if totaltime[m] < mintottime then
                  begin
                     mintottime := totaltime[m];
                     best := m;
                  end;     {if}  {for m}
            if mintottime = maxint
               then
                  begin
   {if no assignment can be made, return error indicator}
                     stat := ER;
                     rawd := ER;
                     writeln(procname,'ER');
                     goto 9999;
```

```
                end      {then}
             else                              {assign process}
                begin
   {selected node is indicated by to location}
   {if file is to be moved, indicate from and to locations}
                   procdest.node := tonode[best];
                   procutil.node := fromnode[best];
                   writeln(procname,fromnode[best],tonode[best]);
                {  procdest.status := ASSIGNED;  }
                   resourceavail[procdest.node] :=
                                 resourceavail[procdest.node] + 1;
                                       {update processor availability}
                   for l := 1 to MAXNODE do      {update filelocation}
                      if l = tonode[best] then
                         filelocation[i,l] := true else
                         filelocation[i,l] := false;   {if}  {for l}
                end;     {else}  {if}
         end;        {if}  {for i}
   stat := OK;
   rawd := OK;
9999: end;        {rawd}
```

APPENDIX D

PASCAL SOURCE CODE FOR THE BASE-LINE ALGORITHM

PASCAL SOURCE CODE FOR THE BASE-LINE ALGORITHM

```
function rawd (t: tgptr; var overhead, stat: integer): integer;
{ Resource allocation and work distribution base-line algorithm
  Files are not moved.
  Each process is run on the node containing its object file. }

label 9999;

var
   i, j: integer;

begin
   overhead := t^.numnodes * 250;
   writeln('time = ',curtime.l,'.',curtime.r);
   for i := 1 to t^.numnodes do with t^.tgentry[i] do
   {consider each process in the task graph in order}
      begin
         j := 1;
         while (j <= MAXNODE) and ((fileavail[j].status <> FREE) or
            (resourceavail[j] = MAXUTIL)) do j := j + 1;
   {select single location of file as execution location}
         if j > MAXNODE then
            begin
   {if no assignment can be made, return error indicator}
               stat := ER;
               rawd := ER;
               writeln(procname,'ER');
               goto 9999;
            end;
   {indicate selected location as to location}
   {since no files are to be moved, from location equals to location}
         procutil.node := j;
         procdest.node := j;
         writeln(procname,j);
      end;
```

```
   stat := OK;
   rawd := OK;
9999: end; { rawd }
```

APPENDIX E

A SAMPLE ASSIGNMENT TRACE

A SAMPLE ASSIGNMENT TRACE

```
START OF SIMULATION
BEGIN EXECUTING EVENTS
time =          1.      26000      {the time of the assignment}
o4               1          4     {the object file is moved from node 4 to node 1}
d4               4                {the input file remains at node 4}
d99              2                {the output file remains at node 2}
time =          1.      27100
o1               1          1
d1               1
d86              3
time =          1.      28500
o4               1          4
d4               4
d79              1
time =          1.      29700
o2               1          2
d2               2
d87              4
time =          1.      31300
o1               1          1
d1               1
d31              3
time =          1.      30300
o42              5          2
d402             2
d417             4
time =          1.      30700
o33              4          3
d303             3
d378             5
time =          1.      31200
o24              3          4
d204             4
d264             1
```

time =	1.	31700
o15	2	2
d105	5	
d175	2	
time =	1.	33200
o4	1	1
d4	4	
d39	1	
time =	1.	33000
o41	5	1
d401	1	
d426	3	

APPENDIX F

AVERAGE USER RESPONSE TIMES

FOR EXPERIMENTS WITH DIFFERENT TYPES OF WORK REQUESTS

AVERAGE USER RESPONSE TIMES
UNI-DIRECTIONAL RING NETWORK

SINGLE FILE COPIES

Case No.	Work Requests	Algorithm 1	Algorithm 2	Algorithm 3	Base-Line Algorithm
1	Type 1	2.7469	3.0103	2.8238	2.6580
		3.0147	3.0323	2.9761	2.8571
		2.9803	3.1979	2.4224	2.7875
	Mean	2.9140	3.0802	2.7408	2.7675
	S.D.	0.1457	0.1026	0.2860	0.1010
2	Type 2	4.2966	4.5874	5.6944	5.1208
		4.5050	5.0819	4.9983	4.7767
		4.5682	4.7541	5.0472	4.7979
	Mean	4.4566	4.8078	5.2305	4.8985
	S.D.	0.1421	0.2516	0.3593	0.1928
3	Type 3	4.9274	6.7356	6.5827	5.3881
		6.8095	4.9597	5.1167	6.0332
		7.5270	5.2643	5.7946	5.2115
	Mean	6.4213	5.6532	5.8313	5.4427
	S.D.	1.3426	0.9497	0.7337	0.4325

AVERAGE USER RESPONSE TIMES
UNI-DIRECTIONAL RING NETWORK

MULTIPLE FILE COPIES

Case No.	Work Requests	Algorithm 1	Algorithm 2	Algorithm 3
4	Type 1	2.5209	2.9166	2.5869
		2.4127	3.0451	2.8721
		2.4246	2.8911	2.7080
	Mean	2.4527	2.9509	2.7223
	S.D.	0.0593	0.0825	0.1431
5	Type 2	3.1635	4.6679	3.4777
		3.0737	4.6429	3.1361
		3.1815	4.4713	3.3009
	Mean	3.1396	4.5940	3.3049
	S.D.	0.0578	0.1070	0.1708
6	Type 3	6.1546	4.1147	4.7966
		7.7251	4.9254	3.4177
		4.6413	5.5236	4.7575
	Mean	6.1737	4.8546	4.3239
	S.D.	1.5420	0.7071	0.7851

AVERAGE USER RESPONSE TIMES
FULLY-CONNECTED NETWORK

SINGLE FILE COPIES

Case No.	Work Requests	Algorithm 1	Algorithm 2	Algorithm 3	Base-Line Algorithm
7	Type 1	2.1700	2.7379	2.2900	2.5144
		2.6142	2.5340	2.3747	2.3993
		2.1613	2.4887	2.4224	2.1760
	Mean	2.3157	2.5869	2.3624	2.3632
	S.D.	0.2590	0.1328	0.0671	0.1721
8	Type 2	2.6966	2.7970	2.9775	2.7168
		2.7810	2.8614	3.0551	2.6345
		2.6770	2.9977	3.1789	2.6220
	Mean	2.7184	2.8854	3.0705	2.6711
	S.D.	0.0553	0.1025	0.1016	0.0419
9	Type 3	4.9458	4.4717	4.5821	5.0674
		3.7665	4.3426	4.0186	3.2802
		4.5400	4.5045	3.8674	3.6643
	Mean	4.4174	4.6929	4.1563	4.0040
	S.D.	0.5991	0.4991	0.3767	0.9408

AVERAGE USER RESPONSE TIMES

FULLY-CONNECTED NETWORK

MULTIPLE FILE COPIES

Case No.	Work Requests	Algorithm 1	Algorithm 2	Algorithm 3
10	Type 1	2.0987	2.3761	2.3924
		2.2238	3.0140	2.5141
		2.0872	2.4467	2.4012
	Mean	2.1366	2.6123	2.4359
	S.D.	0.0758	0.3497	0.0679
11	Type 2	2.6531	2.8505	2.6246
		2.6778	2.9199	2.6696
		2.6907	2.7151	2.7640
	Mean	2.6739	2.8285	2.6861
	S.D.	0.0191	0.1042	0.0711
12	Type 3	4.2410	3.3934	3.5352
		4.1492	4.4629	4.0083
		3.9331	3.9579	4.2930
	Mean	4.1078	3.9381	3.9455
	S.D.	0.1581	0.5350	0.3828

AVERAGE USER RESPONSE TIMES
GLOBAL BUS NETWORK

SINGLE FILE COPIES

Case No.	Work Requests	Algorithm 1	Algorithm 2	Algorithm 3	Base-Line Algorithm
13	Type 1	4.0021	4.8196	3.9222	4.4313
		3.7250	5.0964	3.8986	4.4527
		3.7945	4.9271	3.8112	4.4676
	Mean	3.8405	4.8196	3.9222	4.4505
	S.D.	0.1442	0.3434	0.1244	0.0183
14	Type 2	7.6340	9.8550	8.0955	9.8058
		7.8304	9.3194	7.7541	9.5391
		7.8935	9.5255	8.7272	9.5367
	Mean	7.7860	9.5666	8.1923	9.6272
	S.D.	0.1353	0.2702	0.4937	0.1547
15	Type 3	5.7087	4.7464	5.9313	8.9610
		5.3602	6.6548	5.1269	8.4758
		5.2114	8.1870	5.5117	5.1722
	Mean	5.4268	6.5294	5.5233	7.5363
	S.D.	0.2553	1.7237	0.4023	2.0617

AVERAGE USER RESPONSE TIMES

GLOBAL BUS NETWORK

MULTIPLE FILE COPIES

Case No.	Work Requests	Algorithm 1	Algorithm 2	Algorithm 3
16	Type 1	2.0682	3.8823	2.4484
		2.3330	4.0917	2.2291
		2.1873	4.3497	2.2632
	Mean	2.1962	4.1079	2.3136
	S.D.	0.1326	0.2341	0.1180
17	Type 2	4.2100	6.2170	3.2030
		4.4271	5.3846	3.1824
		4.0106	5.6544	3.0914
	Mean	4.2159	5.7520	3.1589
	S.D.	0.2083	0.4247	0.0594
18	Type 3	5.4290	7.3738	5.5780
		4.7821	7.1361	5.6149
		5.7950	7.1305	5.0858
	Mean	5.3354	7.2135	5.4262
	S.D.	0.5129	0.1389	0.2954

SECTION 9

Abra80 Abraham, S.M., and Dalal, Y.K. Techniques for decentralized management of distributed systems. COMPCON Spring 1980, San Fransisco, California, February 25-28, 1980, 430-437.

Abra73 Abramson, N. The Aloha System. In N. Abramson and F.F. Kuo (Eds.), Computer-Communication Networks. Englewood Cliffs, New Jersey: Prentice-Hall, 1973, 501-517.

Akin78 Akin, T.A., Flinn, P.B., and Forsyth, D.N. A prototype for an advanced command language. Proceedings of the 16th Annual Southeastern Regional ACM Conference, April 1978, 96-102.

Alme79 Almes, G.T., and Lazowska, E.D. The behavior of Ethernet-like computer communications networks. Proceedings of the Seventh Symposium on Operating Systems Principles, December 1979, 66-81.

Amer80 American National Standards Institute. Draft proposal ISO/DP (Data processing - Open systems interconnection - Basic reference model). New York, December 3, 1980.

Amos80 Amoss, J.J. Planning for the Bell Operations System Network. Proceedings of the Fifth International Conference on Computer Communications, Atlanta, Georgia, October 27-30, 1980, 225-230.

Ande75 Anderson, G.A., and Jensen, E.D. Computer interconnection structures: Taxonomy, characteristics, and examples. Computing Surveys, 7, 4 (December 1975), 197-213.

Appl77 Applied Computer Research. Capacity management and software physics. EDP Performance Review, 7, 6 (June 1977), 1-8.

Bach78 Bachman, C.W. Provisional model of open system architecture. Proceedings of the Third Berkeley Workshop on Distributed Data Management and Computer Networks, Berkeley, California, August 29-31,

1978, 1-20.

Bake80 Baker, C.T. Logical distribution of applications and data. IBM Systems Journal, 19, 2 (1980), 171-191.

Bern80 Bernstein, P.A., and Goodman, N. Fundamental algorithms for concurrency control in distributed database systems. Computer Corporation of America Technical Report CCA-80-05, Cambridge, Massachusetts, February 15, 1980.

Bern78 Bernstein, P.A., Rothnie, J.B., Goodman, N., and Papadimitriou, C.A. The concurrency control mechanism of SDD-1: A system for distributed databases (The fully redundant case). IEEE Transactions on Software Engineering, SE-4, 3 (May 1978), 154-168.

Bokh79 Bokhari, S.H. Dual processor scheduling with dynamic reassignment. IEEE Transactions on Software Engineering, SE-5, 4 (July 1979), 341-349.

Bozz80 Bozzetti, M., and Ravasio, P.C. Internetting among local and long haul networks: A case study. Proceedings of the Fifth International Conference on Computer Communications, Atlanta, Georgia, October 27-30, 1980, 729-734.

Brin73 Brinch Hansen, Per. A case study: RC 4000, Operating System Principles. Englewood Cliffs, New Jersey: Prentice-Hall, 1973, Chapter 8.

Brit80 Britton, D.E., and Stickel, M.E. An interprocess communication facility for distributed applications. Proceedings of COMPCON Fall 1980, Washington, D.C., September 23-25, 1980, 590-595.

Brun80 Bruno, J., Jones, J.W., and So, K. Deterministic scheduling with pipelined processors. IEEE Transactions on Computers, C-29, 4 (April 1980), 308-316.

Brya81 Bryant, R.M., and Finkel, R.A. A stable distributed scheduling algorithm. Proceedings of the Second International Conference on

Distributed Computing Systems, Paris, France, April 8-10, 1981, 314-323.

Buck79 Buckles, B.P., and Hardin, D.M. Partitioning and allocation of logical resources in a distributed computing environment. General Research Corporation, 1979.

Caba79a Cabanel, J.P., Marouane, M.N., Besbes, R., Sazbon, R.D., and Diarra, A.K. A decentralized OS model for ARAMIS distributed computer system. Proceedings of the First International Conference on Distributed Computing Systems, Huntsville, Alabama, October 1-5, 1979, 529-535.

Caba79b Cabanel, J.P., Sazbon, R.D., Diarra, A.K., Marouane, M.N., and Besbes, R. A decentralized control method in a distributed system. Proceedings of the First International Conference on Distributed Computing Systems, Huntsville, Alabama, October 1-5, 1979, 651-659.

Case77a Casey, L.(M). Computer structures for distributed systems. Doctoral dissertation, University of Edinburgh, Edinburgh, England, December 1977.

Case77b Casey, L., and Shelness, N. A domain structure for distributed computer systems. Proceedings of the Sixth ACM Symposium on Operating System Principles, November 1977, 101-108.

Case72 Casey, R.G. Allocation of copies of a file in an information network. AFIPS Conference Proceedings, 41, 1 (1972), 617-625.

Chan79a Chandy, K.M., and Misra, J. Distributed simulation: A case study in design and verification of distributed programs. IEEE Transactions on Software Engineering, SE-5, 5 (September 1979), 440-452.

Chan79b Chang, S.R., and Liu, C.N. Modeling and design of distributed information systems. Advances in Computers. New York: Academic Press, 18 (1979), Chapter 3.

Chen80 Chen, P.P., and Akoka, J. Optimal design of distributed information systems. IEEE Transactions on Computers, C-29, 12 (September 1980), 1068-1080.

Chow79 Chow, Y., and Kohler, W.H. Models for dynamic load balancing in a heterogeneous multiple processor system. IEEE Transactions on Computers, C-28, 5 (May 1979), 354-361.

Chu69 Chu, W.W. Optimal file allocation in a multi-computer information system. IEEE Transactions on Computers, C-18 (1969), 885-889.

Chu73 Chu, W.W. Optimal file allocation in a computer network. In N. Abramson and F.F. Kuo (Eds.), Computer-Communication Networks. Englewood Cliffs, New Jersey: Prentice-Hall, 1973, 82-94.

Chu80 Chu, W.W., Holloway, L.J., Lan, M.T., and Efe, K. Task allocation in distributed data processing. Computer, November 1980, 57-69.

Clar80 Clark, D.D., and Svobodova, L. Design of distributed systems supporting local autonomy. Proceedings of COMPCON Spring 1980, San Fransisco, February 25-28, 1980, 438-444.

Crai80 Craig, L.C. Office automation at TI. *Proceedings of COMPCON Fall* 1980, Washington, D.C., September 23-25, 1980, 69-75.

Deci80 Decina, M., and Parodi, R. Circuit and packet-switched data communication in integrated services digital networks. Proceedings of the Fifth International Conference on Computer Communications, Atlanta, Georgia, October 27-30, 1980, 753-759.

Denn78 Denning, P.J. Operating systems principles for data flow networks. Computer, July 1978, 86-96.

DesJ78 desJardins, R., and White, G. ANSI Reference Model for Distributed Systems. Proceedings of the IEEE Computer Society Conference 1978, 144-149.

Dion80 Dion, J. The Cambridge File Server. Operating Systems Review, 14, 4 (October 1980), 26-35.

Eckh78 Eckhouse, R.H., and Stankovic, A. Issues in distributed processing - An overview of two workshops. Computer, January 1978, 22-26.

Elli77 Ellis, C.A. Consistency and correctness of duplicate database systems. Sixth Symposium on Operating System Principles, 1977, 67-84.

Ensl74 Enslow, P.H. (Ed.) Multiprocessors and Parallel Processing. New York: John Wiley and Sons, 1974.

Ensl78 Enslow, P.H. What is a distributed processing system? Computer, January 1978, 13-21.

Ensl81a Enslow, P.H., and Saponas, T.G. Distributed and decentralized control in fully distributed processing systems - A survey of applicable models. Technical Report GIT-ICS-81/02, Georgia Institute of Technology, Atlanta, Georgia, February 1981.

Ensl81b Enslow, P.H., and Saponas, T.G. Performance of distributed and decentralized control models for fully distributed processing systems - Initial simulation studies. Technical Report GIT-ICS-81/09, Georgia Institute of Technology, Atlanta, Georgia, June 1981.

Farb72 Farber, D.J., and Larson, K.C. The system architecture of the Distributed Computer System - The communication system. Proceedings of the Symposium on Computer-Communications Network and Teletraffic, Polytechnic Press, April 1972, 21-27.

Ferr78 Ferrari, D. Computer Systems Performance Evaluation. Englewood Cliffs, New Jersey: Prentice-Hall, 1978, Chapter 1.

Fisc80a Fischer, M.J., Griffeth, N.D., Guibas, L.J., and Lynch, N.A. Optimal placement of identical resources in a distributed network. Technical Report GIT-ICS-80/13, Georgia Institute of Technology, Atlanta, Georgia, October, 1980.

Fisc80b Fischer, M.L., and Hochbaum, D.S. Database location in computer networks. Journal of the ACM, October 1980, 718-735.

Flet80 Fletcher, J.G., and Watson, R.W. Service support in a network operating system. Proceedings of COMPCON Spring 1980, San Fransisco, California, February 25-28, 1980, 344-349.

Fors80 Forsdick, H.C., MacGregor, W.I., Schantz, R.E., and Thomas, R.H. Distributed operating system design study: Phase I. Bolt Beranek and Newman (BBN) Report No. 4455, Cambridge, Massachusetts, August 1980.

Fors78 Forsdick, H.C., Schantz, R.E., and Thomas, R.H. Operating systems for computer networks. Computer, January 1978, 48-57.

Fran73 Frank, H., and Frisch, I.T. Planning computer-communication networks. In N. Abramson and F.F. Kuo (Eds.), Computer-Communication Networks. Englewood Cliffs, New Jersey: Prentice-Hall, 1973, 1-27.

Free75 Freeman, P. Software Systems Principles. Chicago: Science Research Associates, 1975, Chapter 7.

Freu79 Freund, J. E. Modern Elementary Statistics (Fifth Edition). Englewood Cliffs, New Jersey: Prentice-Hall, 1979, Chapter 10.

Garc79a Garcia-Molina, H. Performance update algorithms for replicated data in a distributed database. Doctoral dissertation, Stanford University, June 1979.

Garc79b Garcia-Molina, H. Centralized control update algorithms for fully redundant distributed databases. Proceedings of the First International Conference on Distributed Computing Systems, Huntsville, Alabama, October 1-5, 1979, 699-705.

Gard80 Gardarin, G., and Chu, W.W. A distributed control algorithm for reliably and consistently updating replicated databases. IEEE Transactions on Computers, C-29, 12 (December 1980), 1060-1068.

Gerl80 Gerla, M., and Nilsson, P.O. Routing and flow control interplay in computer networks. Proceedings of the Fifth International Conference on Computer Communications, Atlanta, Georgia, October 1980, 84-89.

Gray78 Gray, J.N. Notes on data base operating systems. In G. Goos and J. Hartmanis (Eds.), Lecture Notes in Computer Science, Number 60, Operating Systems, An Advanced Course, New York: Springer-Verlag, 1978, 393-481.

Gray79 Gray, J.P., and McNeill, T.B. SNA multiple-system networking. IBM Systems Journal, 18, 2 (1979), 263-297.

Hals77 Halstead, M. H. Elements of Software Science. New York: Elsevier North-Holland, 1977.

Hamm80 Hammond, R.A. Experiences with the Series/1 distributed system. Proceedings of COMPCON Fall 1980, Washington, D.C., September 23-25, 1980, 585-589.

Hand79 Handler, G.Y., and Mirchandani, P.B. Location on Networks. Cambridge, Massachusetts: MIT Press, 1979, Chapter 1.

Hirs79 Hirsch, A.R. Using the System Measurement Facilities for performance analysis. CMG Transactions (A newsletter for members of the Computer Measurement Group, Inc.), 23-24, (March-June 1979), 4-2 to 4-6.

Iran79 Irani, K.B., and Khabbaz, N.G. A model for a combined communication network design and file allocation for distributed data bases. Proceedings of the First International Conference on Distributed Computing Systems, Huntsville, Alabama, October 1-5, 1979, 15-21.

Iran81 Irani, K.B., and Khabbaz, N.G. A combined communication network design and file allocation for distributed databases. Proceedings of the Second International Conference on Distributed Computing Systems, Paris, France, April 8-10, 1981, 197-210.

Jens78 Jensen, E.D. The Honeywell Experimental Distributed Processor -An overview. Computer, January 1978, 28-38.

Jone80 Jones, A.K., and Schwarz, P. Experience using multiprocessor systems - A status report. ACM Computing Surveys, 12, 2 (June 1980), 121-165.

Kenn80 Kennington, J.L., and Helgason, R.V. Algorithms for Network Programming. New York: John Wiley and Sons, 1980.

Kerm80 Kermani, P., and Kleinrock, L. A tradeoff study of switching systems in computer communications networks. IEEE Transactions on Computers, C-28, 12 (December 1980), 1052-1060.

Khab80 Khabbaz, N.G. A combined communication network design and file allocation for distributed databases. Doctoral dissertation, University of Michigan, 1980.

Klei73 Kleinrock, L. Scheduling, queueing, and delays in time-shared systems and computer networks. In N. Abramson and F.F. Kuo (Eds.), Computer-Communication Networks. Englewood Cliffs, New Jersey: Prentice-Hall, 1973, 95-141.

Klei80 Kleinrock, L., and Tseng, C.W. Flow control based on limited permit generation rates. Proceedings of the Fifth International Conference on Computer Communications, Atlanta, Georgia, October 1980, 785-790.

Kole77 Kolence, K.W. An Introduction to Software Physics. Palo Alto, California: Institute for Software Engineering, Inc., January 1977.

Kole79 Kolence, K.W. CPU power analysis theory and practice. First International Conference on Computer Capacity Management, Washington, D.C., April 30-May 2, 1979.

Krat80 Kratzer, A., and Hammerstrom, D. A study of load levelling. proceedings of COMPCON Fall 1980, Washington, D.C., September 23-25, 1980, 647-654.

Kuma79 Kumar, B., and Gonsalves, T.A. Modeling and analysis of distributed software systems. Proceedings of the Seventh Symposium on Operating Systems Principles, December 1979, 2-8.

LeBl80 LeBlanc, R.J., Myers, J.T., and Newell, S.R. A simulator for the evaluation of operating system algorithms for fully distributed systems. Unpublished report, Georgia Institute of Technology, Atlanta, Georgia, April 1980.

LeBl81 LeBlanc, R.J., and Maccabe, A.B. PRONET: Language features for distributed programming. Interim Technical Report GIT-ICS-81/03, Georgia Institute of Technology, Atlanta, Georgia, May 1981.

LeLa81 Le Lann, G. A distributed system for real-time transaction proces-

sing. Computer, February 1981, 42-48.

Levi75 Levin, D.K., and Morgan, H.L. Optimizing distributed data bases - A framework for research. AFIPS Conference Proceedings, 44 (1975), 473-478.

Lisk79 Liskov, B. Primitives for distributed computing. Proceedings of the Seventh Symposium on Operating Systems Principles, December 1979, 33-42.

Lync80 Lynch, N.A. Fast allocation of nearby resources in a distributed system. Technical Report GIT-ICS-80/04, Georgia Institute of Technology, Atlanta, Georgia, March 1980.

Lync79 Lynch, N.A., and Fisher, M.J. On describing the behavior and implementation of distributed systems. Unpublished paper, Georgia Institute of Technology, Atlanta, Georgia, May 1979.

Macc80 Maccabe, A.B., and LeBlanc, R.J. A language model for fully distributed systems. proceedings of COMPCON Fall 1980, Washington, D.C., September 23-25, 1980, 723-728.

Mahm76 Mahmoud, S., and Riordan, J.S. Optimal allocation of resources in distributed information networks. ACM Transactions on Database Systems, 1, 1 (March 1976), 66-78.

Mand79 Mandl, C. Applied Network Optimization. New York: Academic Press, 1979.

Mann77 Manning, E.G., and Peebles, R.W. A homogeneous network for data-sharing: Communications. Computer Networks, 1, 4 (1977), 221-224.

Marc81 Marcoliese, R., and Novarese, R. Module and data allocation methods in distributed systems. Proceedings of the Second International Conference on Distributed Computing Systems, Paris, France, April 8-10, 1981, 50-59.

Metc76 Metcalf, R.M., and Boggs, D.R. Ethernet: Distributed packet switching for local computer networks. Communications of the ACM, 19, 7

(July 1976), 395-404.

Mill79 Miller, L.J. Optimal scheduling of task groups on tightly coupled multiprocessors. Doctoral dissertation, State University of New York at Buffalo, February 1979.

Mill76 Mills, D.L. An overview of the Distributed Computer Network. AFIPS Conference Proceedings, 45, (1976), 523-531.

Mont77 Montgomery, W.A. Measurement of sharing in Multics. Proceedings of the Sixth ACM Symposium on Operating Systems Principles, November 1977, 85-90.

Morg77 Morgan, H.L., and Levin, K.D. Optimal program and data location in computer networks. Communications of the ACM, May 1977, 315-321.

Myer80 Myer, T.H. Future message system design: Lessons from the Hermes experience. Proceedings of COMPCON Fall 1980, Washington, D.C., September 23-25, 1980, 76-84.

Need79 Needham, R.M. System aspects of the Cambridge Ring. Proceedings of the Seventh Symposium on Operating Systems Principles, December 1979, 82-85.

Nels80 Nelson, D.L. Application engineering on a distributed computer architecture. Proceedings of COMPCON Spring 1980, San Fransisco, California, February 25-28, 1980, 425-429.

Oust80 Ousterhout, J.K., Scelza, D.A., and Sindhu, P.S. Medusa: An experiment in distributed operating system structure. Communications of the ACM, February 1980, 92-104.

Peeb80 Peebles, R., and Dopriak, T. Adapt: A guest system Proceedings of COMPCON Spring 1980, San Fransisco, California, February 25-28, 1980, 445-454.

Peeb78 Peebles, R., and Manning, E. System architecture for distributed data management. Computer, January 1978, 40-47.

Rao79 Rao, G.S., Stone, H.S., and Hu, T.C. Assignment of tasks in a distributed processor system with limited memory. IEEE Transactions on Computers, C-28, 4 (April 1979), 291-299.

Ritc74 Ritchie, D.M., and Thompson, K. The Unix Time-Sharing System. Communications of the ACM, July 1974, 365-375.

Robe73 Roberts, L.G., and Wessler, B.D. The ARPA network. In N. Abramson and F.F. Kuo (Eds.), Computer-Communication Networks. Englewood Cliffs, New Jersey: Prentice-Hall, 1973, 485-500.

Rose78 Rosenkrantz, D.J., Stearns, R.E., and Lewis, P.M. System level concurrency control for distributed database systems. ACM Transactions on Database Systems, 3, 2 (June 1978), 178-198.

Roth77 Rothnie, J.B., and Goodman, N. An overview of the preliminary design of SDD-1: A system for distributed databases. Computer Corporation of America, Technical Report CCA-77-04, March 1977.

Rowe73 Rowe, L.A., Hopwood, M.D., and Farber, D.J. Software methods for achieving fail-soft behavior in the Distributed Computer System. IEEE Symposium on Computer Software Reliability, April 1973, 7-11.

Salt78 Saltzer, J.H. Naming and binding of objects. In G. Goos and J. Hartmanis (Eds.), Lecture Notes in Computer Science, Number 60, Operating Systems, An Advanced Course. New York: Springer-Verlag, 1978, 99-208.

Samo80 Samoylenko, S.I. Adaptive switching. Proceedings of the Fifth International Conference on Computer Communications, Atlanta, Georgia, October 27-30, 1980, 771-776.

Sapo80 Saponas, T.G., and Crews, P.L. A model for decentralized control in a fully distributed processing system. Proceedings of COMPCON Fall 1980, Washington, D.C., September 23-25, 1980, 307-312.

Sapo81 Saponas, T.G. Distributed and decentralized control in a fully distributed processing system. Doctoral dissertation, Georgia

Institute of Technology, Atlanta, Georgia, December 1981.

Shoc78 Shoch, J.F. Inter-network naming, addressing and routing. Proceedings of the IEEE Computer Society Conference 1978, 72-79.

Shoc80 Shoch, J.F., and Hupp, J.A. Measured performance of an Ethernet local network. Communications of the ACM, December 1980, 711-720.

Sinc80 Sincoskie, W.D., and Farber, D.J. The Series/1 Distributed Operating System: Description and comments. Proceedings of COMPCON Fall 1980, Washington, D.C., September 23-25, 1980, 579-584.

Smit78 Smith, R.G. A framework for problem solving in a distributed processing environment. Stanford Heuristic Programming Project Memo HPP-78-28, Stanford University, December 1978.

Smit80 Smith, R.G. The Contract Net Protocol: High-level communication and control in a distributed problem solver. IEEE Transactions on Computers, C-29, 12 (December 1980), 1104-1113.

Solo79 Solomon, M.H., and Finkel, R.A. The Roscoe distributed operating system. Proceedings of the Seventh Symposium on Operating Systems Principles, December 1979, 108-114.

Ston77 Stone, H.S. Multiprocessor scheduling with the aid of network flow algorithms. IEEE Transactions on Software Engineering, SE-3, 1 (January 1977), 85-93.

Ston78a Stone, H.S. Critical load factors in two-processor distributed systems. IEEE Transactions on Software Engineering, SE-4, 3 (May 1978), 254-258.

Ston78b Stone, H.S., and Bokhari, S.H. Control of Distributed Processes. Computer, July 1978, 97-106.

Syke80 Sykes, D.J. The economics of distributed systems. Proceedings of COMPCON Fall 1980, Washington, D.C., September 23-25, 1980, 8-15.

Thom73 Thomas, R.H. A resource sharing executive for the ARPANET. AFIPS

Conference Proceedings, 42, (June 1973), 155-163.

Thom76 Thomas, R.H. A solution to the update problem for multiple copy data bases which uses distributed control. Bolt Beranek and Newman (BBN) Report No. 3340, 1976.

Thur80 Thurber, K.J. An assessment of the status of network architectures. Proceedings of COMPCON Fall 1980, Washington, D.C., September 23-25, 1980, 87-93.

VonT81 von Tilborg, A.M., and Wittie, L.D. Wave scheduling: Distributed allocation of task forces in network computers. Proceedings of the Second International Conference on Distributed Computing Systems, Paris, France, April 8-10, 1981, 337-347.

Ward80 Ward, S.A. Trix: A network-oriented operating system. Proceedings of COMPCON Spring 1980, San Fransisco, California, February 25-28, 1980, 344-349.

Wilk80 Wilkes, M.V., and Needham, R.M. The Cambridge Model Distributed System. Operating Systems Review, 14, 1 (January 1980), 21-29.

Witt79 Wittie, L.D. A distributed operating system for a reconfigurable network computer. Proceedings of the First International Conference on Distributed Computing Systems, Huntsville, Alabama, October 1-5, 1979, 669-678.

Witt80 Wittie, L.D., and Tilborg, A.M. MICROS, a distributed operating system for MICRONET, a reconfigurable network computer. IEEE Transactions on Computers, C-29, 12 (December 1980), 1133-1144.

Wu80 Wu, S.B., and Liu, M.T. Assignment of tasks and resources for distributed processing. Proceedings of COMPCON Fall 1980, Washington,D.C., September 23-25, 1980, 655-662.

Yeh78 Yeh, R.T., and Chandy, K.M. On the design of elementary distributed systems. Proceedings of the Third Berkeley Workshop on Distributed Data Management and Computer Networks, Berkeley, California, August

29-31, 1978, 289-321.

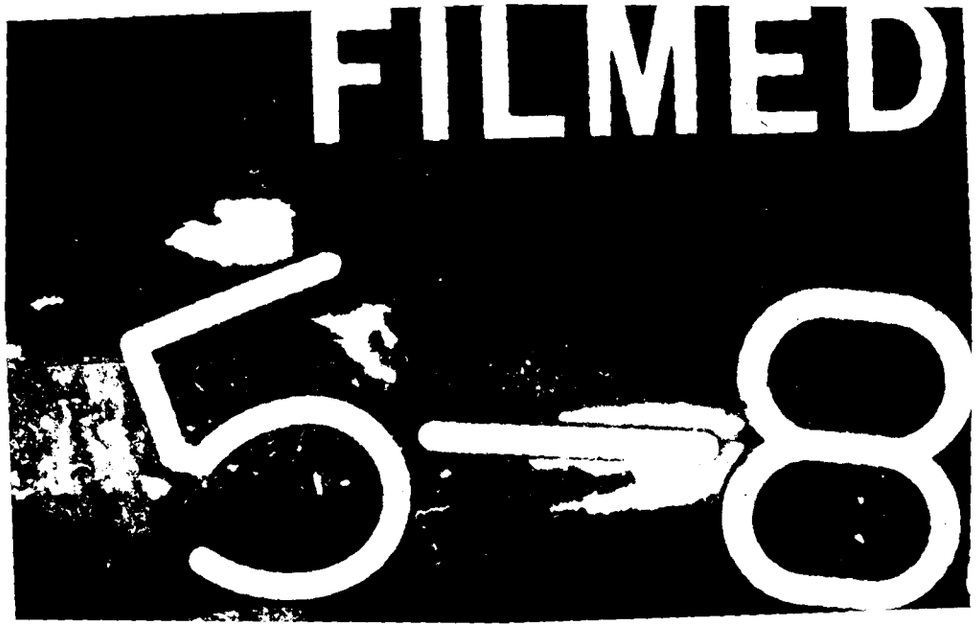
SECTION 10

Donald D. Sharp, Jr. was born in Jonesboro, Arkansas, on May 19, 1938. He grew up in Memphis, Tennessee and graduated from Central High School there. In 1960, he received a bachelor's degree from Rice University, where he majored in Electrical Engineering. After serving for two years as an officer in the U. S. Navy, he attended the Wharton School of Finance and Commerce at the University of Pennsylvania. His major at Wharton was in Industrial Management and his minor was in Operations Research and Management Science. He received a Master of Business Administration Degree from Wharton in 1964. His master's thesis was entitled "The use of real-time computers for inventory control".

Mr. Sharp then worked for 16 years in the data processing industry before returning to school to get his doctorate. His first position was as a Systems Engineer with IBM, where he assisted customers with the installation of their computer systems, taught customer classes at the IBM Education Center, and worked in the IBM Test Center. His next position was with a software development and data processing consulting firm, Management Science America (MSA). At MSA, Mr. Sharp designed, programmed, and installed software systems for many businesses and government agencies. In 1970, Mr. Sharp and six other men left MSA and formed their own data processing consulting firm, Consultec, Inc. Mr. Sharp was a partner with that firm and also served as Corporate Treasurer.

Since 1978, Mr. Sharp has attended the School of Information and Computer Science at the Georgia Institute of Technology, where he taught a class in data base design and worked as a Research Technologist and as a Computer Operations Supervisor. He received a Master of Science Degree in Information and Computer Science from Georgia Tech in December of 1980. He will receive his Doctor of Philosophy Degree from Georgia Tech in January of 1982, and then will join the faculty of the Computer Science Department at North Texas State University.

Mr. Sharp is a member of the Association of Computing Machinery and the IEEE Computer Society. He is married and is the father of three children.


FILMED
5-8